

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。

提供超过110种有效构建和运行OpenStack云计算、存储、网络和自动化的方法

OpenStack

云计算实战手册 (第3版)

OpenStack Cloud Computing Cookbook
Third Edition

[英] 凯文·杰克逊 (Kevin Jackson)

[美] 科迪·邦奇 (Cody Bunch) 著

[美] 埃格尔·西格勒 (Egle Sigler)

宋秉金 黄凯 杜玉杰 译



OpenStack

云计算实战手册（第3版）

〔英〕凯文·杰克逊（Kevin Jackson）

〔美〕科迪·邦奇（Cody Bunch） 著

〔美〕埃格尔·西格勒（Egle Sigler）

宋秉金 黄凯 杜玉杰 译

人民邮电出版社

北 京

图书在版编目 (C I P) 数据

OpenStack云计算实战手册：第3版 / (英) 凯文·杰克逊 (Kevin Jackson), (美) 科迪·邦奇 (Cody Bunch), (美) 埃格尔·西格勒 (Egle Sigler) 著; 宋秉金, 黄凯, 杜玉杰译. -- 2版. -- 北京: 人民邮电出版社, 2018.2

ISBN 978-7-115-47242-7

I. ①O… II. ①凯… ②科… ③埃… ④宋… ⑤黄…
⑥杜… III. ①云计算—手册 IV. ①TP393.027-62

中国版本图书馆CIP数据核字(2017)第316952号

版权声明

Copyright © 2015 Packt Publishing. First published in English under the title *OpenStack Cloud Computing Cookbook, Third Edition*.

All rights reserved.

本书由英国 Packt Publishing 公司授权人民邮电出版社出版。未经出版者书面许可, 对本书的任何部分不得以任何方式或任何手段复制和传播。

版权所有, 侵权必究。

-
- ◆ 著 [英]凯文·杰克逊 (Kevin Jackson)
[美]科迪·邦奇 (Cody Bunch)
[美]埃格尔·西格勒 (Egle Sigler)
- 译 宋秉金 黄凯 杜玉杰
- 责任编辑 杨海玲
- 责任印制 焦志炜
- ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京鑫正大印刷有限公司印刷
- ◆ 开本: 800×1000 1/16
印张: 21.25
字数: 415 千字 2018 年 2 月第 2 版
印数: 5 001—7 400 册 2018 年 2 月北京第 1 次印刷
著作权合同登记号 图字: 01-2015-8788 号
-

定价: 69.00 元

读者服务热线: (010)81055410 印装质量热线: (010)81055316

反盗版热线: (010)81055315

广告经营许可证: 京东工商广登字 20170147 号

内容提要

OpenStack 是一个用于构建公有云和私有云的开源软件。本书全面讲解 OpenStack 的方方面面，每一章均提供每种服务的真实且实用的示例，使读者能使用和实践 OpenStack 的最新特性，旨在帮助读者快速上手 OpenStack，在理解的基础上将 OpenStack 应用到自己的数据中心。

本书涵盖了安装和配置一个私有云的各种内容：如何安装和配置 OpenStack 的所有核心组件，并运行一个可管理和可运维的环境；如何掌握一个完整的私有云软件栈，从计算资源的扩容到管理高冗余、高可用的对象储存服务。这一版除了对原有章节内容进行了更新和优化，还新增了关于 OpenStack 网络服务的全面介绍，让读者了解如何对整个云基础设施进行精细化控制。

本书适合熟悉云计算平台并正在从虚拟化环境过渡到云计算环境的系统管理员和架构师阅读。

译者简介

宋秉金 硕士，目前在国内领先的云计算公司腾讯云工作，专注开发者运营及传播云计算最新趋势，致力于帮助中小开发者快速上云。他毕业于北京外国语大学高级翻译学院，有着丰富的技术翻译经验；Python 学习型社区“编程派”的发起人，通过同名公众号与数万名 Python 编程爱好者分享技术经验、传播最新技术资讯。他也是《Python 参考手册（第四版·修订版）》的审校者。

黄凯 有十几年的 IT 行业工作经验，是一名技术跨度很广的 IT 专家，大部分时间使用各种语言在编程或者撰写技术资料，有着企业级数据中心、SOA、语义网、编译系统、分布式计算和保险金融业务系统方面的技术背景和专利，并通过 EMC、VMware、IBM、微软等多项认证。目前，他在 IBM x86 软件团队担任云计算解决方案资深架构师，研究私有/混合/公有云计算环境的管理运维和开源云平台的应用创新工作，并主持撰写了 OpenStack 企业参考架构红皮书。过往的工作经历包括在一家全球领先的数据存储公司担任首席工程师，以及在一家中间件平台公司担任虚拟服务器架构师。

黄凯拥有应用数学的学士学位和软件工程的硕士学位，他经常应邀在一些技术博客上撰文或作为创业项目的评委，并不定期在他的微博 www.weibo.com/topkai 上发表关于云计算技术的见解和点评。

杜玉杰 硕士，开源顾问，中国 OpenStack 社区（COSUG）发起人，关注社区运营和商务拓展相关方向，先后为 IBM、HP、EMC、VMWare 等企业提供开源相关咨询服务，目前担任 OpenStack 基金会董事，企业级云计算联盟（ECA）副秘书长、华为开源顾问、红帽 OpenStack 认证培训讲师、HP 培训部兼职讲师，曾为北航云计算硕士班、上海交大移动云计算硕士班授课。可以通过微博“@ben_杜玉杰”交流开源相关话题。

作者简介

凯文·杰克逊（Kevin Jackson）是一名经验老道的信息技术（IT）专业人士，目前在 Rackspace 作为 OpenStack 及私有云的专家，为各种规模的企业提供服务。他从 2011 年初就开始接触 OpenStack，拥有各种 Linux 和 Unix 操作系统方面的丰富经验。他的 Twitter 账号是@itarchitectkev。

他是本书第 1 版的作者，合著了第 2 版。他还在加利福尼亚一次为期 5 天的快速写书活动中与人合著了 OpenStack 基金会的《OpenStack 架构设计指南》。

我要感谢科迪担任起更新本书的艰巨任务。还要感谢埃格尔对本书出版的支持。本书的技术审稿人来自全球各地，帮助我们成功地达到了更新本书的目的。

我还要感谢我的家人，尽管我不确定他们意识到我又写了一本书。我想这次应该没人会再抱怨我了。

最后，我要感谢 Rackspace 给我机会，并支持我写这本书；还要感谢我打扰的许多朋友，感谢他们耐心回答我的问题。

科迪·邦奇（Cody Bunch）是 Rackspace 主机业务的一名私有云架构师。他从 2012 年开始接触 OpenStack，合著了本书的第 2 版，以及《OpenStack 架构设计指南》。他在 IT 行业有 15 年的从业经验，从事过 SaaS、VoIP、虚拟化等方面的工作。他的 Twitter 账号是@cody_bunch。

我要感谢凯文再次和我一起完成这本书。还要感谢埃格尔不遗余力地让本书变得更好。如果没有众多审稿人及 Packt 出版社同事的支持，就不会有本书的最新版问世。

接下来，更重要的是，我要感谢我的孩子们和太太。没有他们的支持，我无法保证能够及时完稿。同样，还需要感谢我的雇主 Rackspace 的支持与理解。

最后，我要感谢所有的作者、出版社、审稿人及我的老板。虽然参与本版编写和出版工作的人并不多，我想还是要感谢 OpenStack 社区，他们是本书的理想读者人群。你们不仅给予了支持和技术指导，还是新版上市的重要理由。感谢大家！

埃格尔·西格勒（Egle Sigler）是 OpenStack 基金会的董事，还是 Rackspace 私有云团队的一名架构师。她拥有计算机科学硕士学位。她的第一份工作是软件开发，并且仍对那些编写、测试和部署代码的人感到十分亲切，因为在过去的职业生涯中她也亲身经历过所有这些工作。埃格尔梦想有一天编写、测试、部署代码将会成为一个无缝、轻松的过程，不会出现 bug 和令人绝望的情况。埃格尔相信知识应该共享，参与本书的编写就是一种尝试。另外她还积极演讲，组织大会上的研讨会，写博客。埃格尔的 Twitter 账号是@eglute。

她与人合著了《DevOps 实战：VMware 管理员运维方法、工具及最佳实践》。

我要感谢我的丈夫对我写作本书的坚定支持，他不仅是我的爱人，还是我的技术顾问，对本书提出了许多建议。由于某些原因，网络部分的章节需要重点排错。

我希望朋友和家人能够谅解，在写作本书时无法联系到我。

还要感谢所有的 OpenStack 开发者、质量工程师、运营商、用户以及文档作者，感谢你们让 OpenStack 不断完善。

感谢凯文和科迪让我加入本书的写作。我不敢相信本书之前已经拥有如此优秀的内容，以及那些便利的 Vagrant 环境脚本。感谢技术审稿人自愿投入数百个小时进行细致的审校。感谢 Packt 出版社的审稿人和编辑及时地沟通和反馈。感谢 Rackspace 的各位同事提供建议和指导。最后，感谢 Rackspace 支持我参与本书写作。

审稿人简介

克里斯·贝蒂 (Chris Beatty) 是一位专业 IT 人员，在系统管理、基础架构方面有着丰富的经验。他目前在 Rackspace 工作，帮助企业客户设计、运行高性能的企业解决方案。

我想要感谢我的妻子和孩子，让我花时间进行本书的审稿；还要感谢同事请我参与审稿。

沃尔特·本特利 (Walter Bentley) 是 Rackspace 公司的私有云解决方案架构师。他是公司的新员工，在生产系统管理和解决方案架构方面有着广泛的涉猎。他有着 17 年的工作经验，参与过不同行业的 IT 系统设计，包括在线营销、财务、保险、航空、食品和教育行业。过去他一直为使用 OpenStack 等技术的公司提供咨询服务。现在，他是 OpenStack 技术的倡导者，同时从事云教育。

我由衷地感谢作者邀请我参与这本优秀图书的出版工作。

维多利亚·马丁内斯·克鲁兹 (Victoria Martinez de la Cruz) 是阿根廷苏尔国立大学计算机科学与工程学院的计算机科学硕士。在大学的最后几年，她通过 GNOME 的推广项目以及谷歌的编程之夏实习接触了 OpenStack。她目前是 Red Hat 公司的一名软件工程师，OpenStack Trove 和 Zaqr 项目的核心成员。她的主要兴趣是操作系统、网络和数据库。她还热衷于自由和开源软件，喜欢帮助新人参与开源项目。你可通过 victoria@vmartinezdelacruz.com 联系维多利亚。

我要感谢《OpenStack 云计算实战手册（第3版）》的作者及出版社给我机会成为本书的技术审稿人。很难得的经历！

斯蒂芬·伦兹（Stefan Lenz）在慕尼黑的宝马公司工作。他是宝马全球 IT 组织的数据中心及云服务部门的一位经理，负责为全球的宝马业务交付计算、存储和网络服务。

他拥有德国埃朗根大学的核物理学博士学位，并在耶鲁大学从事过博士后研究，利用高性能计算机进行核物理研究。在成为高性能计算机 IT 架构师、加入宝马之前，他曾是德国汽车行业的高性能计算顾问。从 2002 年到 2014 年，他参与了多个整合宝马 IT 组织的计划和项目。

他目前与家人居住在慕尼黑，喜欢在阿尔卑斯山滑雪、徒步和骑行。他与妻子一起写了六本有关徒步、山地骑行和西班牙圣地亚哥朝圣之路的图书。读者可以通过 Twitter 关注 @stefan_km_lenz，或通过他的网站 www.serverfabrik.de 联系他。

2014 年夏，我在家里地下室中废寝忘食地学习 OpenStack 知识。当时使用的就是本书的第 1 版。我要感谢作者给我的巨大帮助。我还要感谢妻子的支持和耐心，感谢她给我的私人实验室捐献了两台她公司的旧计算机。

安迪·麦克雷（Andy McCrae）是 Rackspace 私有云团队的一名软件工程师。安迪于 2007 年参加工作，从伦敦大学学院拿到计算机科学与工程硕士学位之后，他成为了 Rackspace 的一名 Linux 系统管理员。

安迪精通 Swift（对象管理）和 Ansible。他还曾是 OpenStack Chef 项目的核心贡献者，现在正在从事 OpenStack 中 Ansible 部署相关的社区项目。

近期，他还在温哥华 OpenStack 峰会上演讲，分享如何管理 OpenStack 环境中的日志。

梅利莎·帕尔默（Melissa Palmer）是一名系统工程师和架构师，热衷于虚拟化、基础设施和 OpenStack。她拥有工程学士和硕士学位，主攻电力工程和安全网络系统设计。她是 VMUG 成员，积极倡导社区发展，参与过多个 IT 架构和社区项目的专家讨论和播客节目。她还是虚拟设计大师挑战（Virtual Design Master Challenge）的创意总监，网址是 <http://virtualdesignmaster.com>。梅利莎业余喜欢烹饪、写作及观看火箭发射。她的 Twitter 账号是 @vMiss33，还可通过博客 <http://vMiss.net> 联系她。

希拉姆·拉詹（Sirram Rajan）是 Rackspace 的高级工程师，负责为客户设计解决方案并协助自动化实施。加入 Rackspace 之前，他曾是得克萨斯州立大学的系统程序员，在那里拿到了计算机科学的硕士学位。他还具备十余年的专业经验，精通 Linux 系统、网络、编程和安全。在业余时间，他喜欢旅行，动手自创家庭自动化，观看曲棍球比赛，写写有趣的程序，以及与人讨论技术话题。

序

在欧洲核子研究会（CERN），物理学家们和工程师们致力于探究宇宙的基本结构。他们使用世界上最大最复杂的科学仪器，研究物质的基本构成——基本粒子。他们让基本粒子以接近光的速度进行碰撞，碰撞的过程能够帮助物理学家理解原子内部发生了什么，并探索自然界的根本法则。

大型强子对撞机（LHC）是全球规模最大、性能最强的粒子加速器，由 27 千米长的超导磁铁以及许多加速结构组成，用于提升粒子的能量。在加速器中，两个高能粒子束在碰撞之前，以接近光速行进。这里每年都会产生 27 PB 的数据，由 CERN 数据中心的数千台计算机记录并进行分析。

2015 年大型强子对撞机进行了升级，碰撞产生的能量几乎翻倍。很明显科学家们需要更多的计算资源。为了提供额外的资源，更快地响应用户的需求，CERN 采取了一种新的策略。2012 年，CERN 的一个小团队开始研究如何利用 OpenStack 这个开源软件构建云计算。这是一个非常新颖的技术，社区也十分热情，但是复杂度很高。由于当时代码还较新，相应的文档和培训材料缺乏。我们给快速滑流人员启动项目，因此要寻找提高管理效率的途径。这时我们发现了《OpenStack 实践手册》的第四版。它成为了团队新人加入后必看的标准文档，以便帮助他们快速了解相关概念，搭建第一个云环境，然后再投入 CERN 云环境进行工作。

随着云环境不断扩展，OpenStack 慢慢成熟，开发团队设置一直在变化，我们仍继续使用本书作为指南，构建小型云环境来尝试新的概念，理解云计算的复杂性。

多年来，我们参加在 OpenStack 峰会上与社区、科学家和运维交流。由于 OpenStack 发展迅速，我们必须使用本书的最新版本，作者们也做到了持续更新。

序

在欧洲核子研究会（CERN），物理学家们和工程师们致力于探究宇宙的基本结构。他们使用世界上最大最复杂的科学仪器，研究物质的基本构成——基本粒子。他们让基本粒子以接近光的速度进行碰撞。碰撞的过程能够帮助物理学家理解原子间如何交互，从而探索自然界的基本法则。

大型强子对撞机（LHC）是全球规模最大、性能最强的粒子加速器，由 27 千米长的超导磁铁以及许多加速结构组成，用于提升粒子的能量。在加速器中，两个高能粒子光束在碰撞之前，以接近光速行进。这里每年会产生 27 PB 的数据，由 CERN 数据中心的数千台计算机记录并进行分析。

2015 年大型强子对撞机进行了升级，碰撞产生的能量几乎翻倍。很明显科学家们需要更多的计算资源。为了提供额外的资源，更快地响应用户的需求，CERN 采取了一种新的策略。2012 年，CERN 的一个小团队开始研究如何利用 OpenStack 这个开源软件搭建云计算。这是一个非常有前途的技术，社区也十分热情，但是复杂度很高。由于当时代码还较新，相应的文档和培训比较缺乏。我们想快速培训人员启动项目，因此要寻找提高管理员效率的指南。这时我们发现了《OpenStack 实战手册》的第 1 版。它成为了团队新人加入后必看的标准文档，以便帮助他们快速了解相关概念，搭建第一个云环境，然后再投入 CERN 云环境进行工作。

随着云环境不断研究，OpenStack 慢慢成熟，即使团队成员一直在变化，我们仍继续使用本书作为指导，构建小型云环境来尝试新的概念，调研云计算的灵活性。

多年来，我们经常在 OpenStack 峰会上与凯文、科迪和埃格尔交流。由于 OpenStack 发展迅速，有必要使用本书的最新版本，作者们也做到了持续更新。

CERN 的云环境现在拥有两个数据中心，分别位于日内瓦和布达佩斯特，由 3000 台服务器组成，运行了数万个虚拟机。这里经常会有新成员加入，我们将继续使用本书作为团队培训的关键内容，同时期待最新版中更新的内容。

蒂姆·贝尔 (Tim Bell)

基础设施经理, CERN

序

前言

OpenStack 是一个用于构建公有云和私有云的开源软件。它是一个全球性的成功软件，由全球数千名人员开发和支持，并得到当今云计算领域巨头的鼎力支持。本书设计的初衷在于帮助读者快速上手 OpenStack，在理解的基础上将 OpenStack 更有信心地应用到自己的数据中心。本书涵盖了安装和配置私有云的各种内容：从使用 Virtualbox 安装 OpenStack 测试环境，到快速扩张生产环境的自动安装脚本。本书主要包括：

- ❑ 如何安装和配置 OpenStack 的所有核心组件，并运行一个类似 Rackspace、HPHelion 或其他云平台那样可管理和可运维的环境；
- ❑ 如何掌握一个完整的私有云技术栈，从计算资源的扩容到管理高冗余、高可用的对象存储服务；
- ❑ 每一章均提供了各种服务的真实和实用的例子，使读者在应用到自己的环境中时也能充满信心。

本书为成功安装和运行读者自己的私有云提供了一个清晰的按部就班的指导。它包含丰富而实用的脚本，使读者能使用和实践 OpenStack 的最新特性。

本书涵盖的内容

第 1 章带领读者安装和配置 Keystone，它支撑着其他 OpenStack 服务。

第 2 章介绍如何安装、配置和使用 OpenStack 环境中的镜像服务 Glance。

第 3 章介绍如何安装、配置 OpenStack 网络服务，包括 DVR 等新特性。

第 4 章介绍如何配置和使用 OpenStack Nova，并以 Virtualbox 环境运行 OpenStack 计算服务为例进行说明。

第 5 章介绍如何配置和使用 OpenStack 对象存储，并以 Virtualbox 环境运行对象存储服务为例进行说明。

第 6 章介绍如何使用存储服务来存储、获取文件和对象。

第 7 章介绍如何在数据中心中使用可用于运行 OpenStack 存储服务的工具和技巧。

第 8 章介绍如何使用 Cinder 卷服务，安装和配置持久化块存储服务。

第 9 章介绍 OpenStack Neutron 中的 LBaaS、FWaaS 服务，以及 Ceilometer 和 Heat 等 OpenStack 特性。

第 10 章介绍如何使用 Openstack Dashboard，安装和使用 Web 用户界面，执行如创建用户、修改安全组、启动实例等任务。

第 11 章介绍如何使用 Ansible 进行自动化安装，并说明提升 OpenStack 服务弹性和可用性的工具与技巧。

阅读本书需要的准备

要使用本书，需要能够访问带有虚拟化能力的计算机或服务器。要建立一个小型的实验环境，需要一个控制主机、网络主机和计算主机。为了运行 Swift 服务，本书提供了创建多节点环境的具体步骤，环境由 1 个代理服务器和 5 个存储节点组成。

为了设置好实验环境，你需要安装和使用 Oracle 的 Virtualbox 和 Vagrant。可访问 <http://bit.ly/OpenStackCookbookSandbox>，了解如何在计算机上使用 Virtualbox 和 Vagrant。

我们还提供了其他快速上手实验环境的说明，访问地址 <http://www.openstackcookbook.com>。安装 MariaDB/MySQL 等支撑性软件时，请参考该网站的安装说明。更多信息，请访问 <http://bit.ly/OpenStackCookbookPreReqs>。

本书的目标读者

本书面向的是那些熟悉云计算平台，并正在从虚拟化环境过渡到云计算环境的系统管理员和技术架构师。读者需要有虚拟化和 Linux 管理的知识，如果具有 OpenStack 方面的

知识和经验，对阅读本书是有帮助的，但不是必须的。

书中的排版约定

读者会发现在本书中使用了一些文本格式用以区分不同类型的信息。这里介绍一下这些格式的一些例子和含义。

文中的代码、数据库表名、文件夹名称、文件名、文件的扩展名、路径和用户输入等都采用等宽字体表示。

代码块表示方式如下：

```
account-server:bind_port=6000
container-server:bind_port=6001
object-server:bind_port=6002
```

对于代码块中希望重点关注的部分，会将相关内容加粗：

```
[swift-hash]
# Random unique string used on all nodes
swift_hash_path_prefix=a4rUmUIgJYXpKhbh
swift_hash_path_suffix=NESuuUEqc6OXwy6X
```

命令行输入和输出都是以如下样式书写的：

```
sudo swift-init all start
sudo swift-init all stop
sudo swift-init all restart
```

新术语和重要词语用粗体表示。以页面、菜单和对话框中出现的词语为例，读者会看到这样的样式：“一个重要字段是 **Common Name** 字段。”



警告或重要注释会使用这样的段落。



提示和技巧会使用这样的段落。

读者反馈

我们一直非常欢迎读者的反馈。请告诉我们你觉得这本书怎么样——喜欢哪些内容，

不喜欢哪些内容。你的反馈对我们至关重要，能让我们写出使读者获益更多的书。

普通的反馈只需发送邮件到 feedback@packtpub.com，并在邮件标题中注明相应的书名即可。

如果你是某个领域的专业人士，并且对写作和撰稿感兴趣，请参考我们的作者指南，相关网址：www.packtpub.com/authors。

客户支持

现在你已经成为 Packt 图书的尊贵所有者，我们将极尽所能让你能从购买的本书中获得极大的价值。

下载示例代码

读者可从 <https://github.com/OpenStackCookbook/OpenStackCookbook> 下载本书的示例代码文件。所有相关文件均可从此处下载。

我们还提供了其他快速上手实验，请访问 www.packtpub.com 安装 MariaDB、MySQL、MongoDB 等数据库。请参考该网站的安装说明。更多信息，请访问 <http://bit.ly/OpenStackCookbook>。

本书的目标读者

本书面向的是那些熟悉云计算平台，并正在从运维人员过渡到云计算环境的人员。本书旨在帮助你了解 OpenStack 的部署和配置，以及如何利用 OpenStack 来部署和管理应用程序。

目录

第 1 章 Keystone——OpenStack 身份认证服务	1
1.1 简介	1
1.2 安装 OpenStack 身份认证服务	2
1.3 为 SSL 通信配置 OpenStack 身份认证	5
1.4 在 Keystone 里创建租户	6
1.5 在 Keystone 里配置角色	8
1.6 往 Keystone 里添加用户	9
1.7 定义服务端点	13
1.8 创建服务的租户和服务的用户	18
1.9 为 LDAP 的集成配置 Openstack 身份认证	22
第 2 章 Glance——OpenStack 镜像服务	25
2.1 简介	25
2.2 安装 OpenStack 镜像服务	26
2.3 用 OpenStack 身份认证服务配置 OpenStack 镜像服务	29
2.4 用 OpenStack 对象存储配置 OpenStack 镜像服务	30
2.5 用 OpenStack 镜像服务管理镜像	31
2.6 注册远程存储的镜像	35
2.7 租户间共享镜像	36

2.8 查看共享镜像	37
2.9 使用镜像元数据	38
2.10 迁移 VMware 镜像	41
2.11 创建 OpenStack 镜像	42
第 3 章 Neutron——OpenStack 网络服务	48
3.1 简介	48
3.2 在专属网络节点安装 Neutron 和 Open vSwitch	50
3.3 配置 Neutron 和 Open vSwitch	52
3.4 安装并配置 Neutron API 服务	58
3.5 创建租户 Neutron 网络	63
3.6 删除 Neutron 网络	65
3.7 创建外部浮动 IP Neutron 网络	67
3.8 Neutron 网络的不同用途	72
3.9 配置分布式虚拟路由	76
3.10 使用分布式虚拟路由器	81
第 4 章 Nova——OpenStack 计算服务	84
4.1 简介	85
4.2 安装 OpenStack 计算控制节点服务	86
4.3 安装 OpenStack 计算软件包	87
4.4 配置数据库服务	89
4.5 配置 OpenStack 计算服务	90
4.6 使用 OpenStack 身份认证服务配置计算服务	95
4.7 停止和启动 Nova 服务	97
4.8 在 Ubuntu 上安装命令行工具	99
4.9 通过 HTTPS 使用命令行工具	99
4.10 检查 OpenStack 计算服务	101
4.11 使用 OpenStack 计算服务	103

4.12 管理安全组	105
4.13 创建和管理密钥对	107
4.14 启动第一个云实例	109
4.15 修复出错的实例部署	113
4.16 终止实例	115
4.17 使用在线迁移	116
4.18 使用 nova-scheduler	117
4.19 创建实例类型	119
4.20 定义主机分组	120
4.21 在特定可用区启动实例	124
4.22 在特定计算主机启动实例	126
4.23 从集群移除 Nova 节点	127
第 5 章 Swift——OpenStack 对象存储	131
5.1 简介	131
5.2 在 Keystone 中配置 Swift 服务和用户	132
5.3 安装 OpenStack 对象存储服务——代理服务器	134
5.4 配置 OpenStack 对象存储服务——代理服务器	136
5.5 安装 OpenStack 对象存储服务——存储节点	138
5.6 配置 Swift 使用物理存储	139
5.7 配置对象存储备份	141
5.8 配置 OpenStack 对象存储——存储服务	143
5.9 制作对象存储环	145
5.10 停止和启动 OpenStack 对象存储	148
5.11 配置 SSL 访问	149
第 6 章 使用 OpenStack 对象存储	152
6.1 简介	152
6.2 安装 swift 客户端工具	152

6.3	创建容器	154
6.4	上传对象	155
6.5	上传大对象	156
6.6	列出容器和对象	159
6.7	下载对象	160
6.8	删除容器和对象	162
6.9	使用 OpenStack 对象存储访问控制列表	164
6.10	两个 Swift 集群间进行容器同步	166
第 7 章	管理 OpenStack 对象存储	169
7.1	简介	169
7.2	用 swift-init 管理 OpenStack 对象存储集群	169
7.3	检查集群健康状况	171
7.4	管理 Swift 集群容量	173
7.5	从集群中删除节点	177
7.6	检测和更换故障硬盘	178
7.7	收集使用情况统计数据	180
第 8 章	Cinder——OpenStack 块存储	183
8.1	简介	183
8.2	配置 Cinder 卷服务	184
8.3	为 Cinder 卷配置 OpenStack 计算服务	186
8.4	创建卷	189
8.5	为实例添加卷	191
8.6	从实例中分离卷	193
8.7	删除卷	194
8.8	配置第三方卷服务	195
8.9	使用 Cinder 快照	196
8.10	从卷启动	198

第 9 章 深入 OpenStack	200
9.1 简介	200
9.2 使用 cloud-init 运行安装后的命令	200
9.3 使用 cloud-init 运行安装后的配置	202
9.4 安装 OpenStack Telemetry	205
9.5 使用 OpenStack Telemetry 查看使用数据	209
9.6 安装 Neutron LBaaS	213
9.7 使用 Neutron LBaaS	215
9.8 配置 Neutron FWaaS	219
9.9 使用 Neutron FWaaS	222
9.10 安装 OpenStack 编排服务 Heat	228
9.11 使用 Heat 启动实例	231
第 10 章 使用 OpenStack Dashboard	235
10.1 简介	235
10.2 安装 OpenStack Dashboard	236
10.3 使用 OpenStack Dashboard 进行密钥管理	237
10.4 使用 OpenStack Dashboard 管理 Neutron 网络	242
10.5 使用 OpenStack Dashboard 进行安全组管理	248
10.6 使用 OpenStack Dashboard 启动实例	254
10.7 使用 OpenStack Dashboard 终止实例	257
10.8 使用 OpenStack Dashboard 连接到使用 VNC 的实例	258
10.9 使用 OpenStack Dashboard 添加新租户	260
10.10 使用 OpenStack Dashboard 进行用户管理	263
10.11 使用 OpenStack Dashboard 操作 LBaaS	268
10.12 使用 OpenStack Dashboard 进行 OpenStack 编排	276
第 11 章 生产环境中的 OpenStack	285
11.1 简介	285

11.2	安装 MariaDB Galera 集群	286
11.3	MariaDB Galera 集群配置 HA Proxy	288
11.4	配置 HA Proxy 实现高可用	290
11.5	使用 Corosync 安装并配置 Pacemaker	295
11.6	使用 Pacemaker 和 Corosync 配置 OpenStack 服务	299
11.7	绑定多个网卡实现高冗余	304
11.8	使用 Ansible 自动安装 OpenStack——主机配置	305
11.9	使用 Ansible 自动安装 OpenStack——Playbook 配置	309
11.10	使用 Ansible 自动安装 OpenStack——运行 Playbook	315

第 1 章

Keystone——OpenStack 身份认证服务

本章将讲述以下内容：

- ☐ 安装 OpenStack 身份认证服务
- ☐ 为 SSL 通信配置 OpenStack 身份认证
- ☐ 在 Keystone 里创建租户
- ☐ 在 Keystone 里配置角色
- ☐ 往 Keystone 里添加用户
- ☐ 定义服务端点
- ☐ 创建服务对应的租户和用户
- ☐ 为 LDAP 的集成配置 Openstack 身份认证

1.1 简介

OpenStack 身份认证服务 (Identity Service)，即 **Keystone**，是为 OpenStack 云环境中用户的账户和角色信息提供认证和管理服务的。这是一个关键的服务，OpenStack 云环境中所有服务之间的鉴权和认证都需要经过它，所以它也是 OpenStack 环境中第一个要安装的服务。OpenStack 身份认证服务通过在所有 OpenStack 服务之间传输有效的鉴权密钥，来对用户和租户鉴权。这个密钥被用来为某个具体服务做鉴权和验证，接下来你就能使用那些服务，如 OpenStack 的存储和计算服务。因此，第一步就要配置 OpenStack 身份认证服务，包括为用户创建合适的角色，以及创建服务、租户、用户账户和服务 API 端点，这些服务构成了我们的云基础设施。

在 Keystone 里，我们有如下概念：**租户、角色和用户**。租户就像一个项目，它有一些资

源，比如用户、镜像和实例，并且其中有仅仅对该项目可知的网络。用户可隶属于一个或多个租户，并且可以在这些项目中切换，去获取相应资源。租户里的用户可以被指定为多种角色。在最基本的应用场景里，一个用户可以被指定为管理员角色，或者只是成员角色。当用户在租户中拥有管理员特权时，他们可以使用那些影响租户的功能（比如修改外部网络）。反之，一个普通用户被指定为成员角色，它通常被指定执行与用户相关的角色，比如旋转实例、创建卷和创建租户唯一网络。

1.2 安装 OpenStack 身份认证服务

我们将会使用 Ubuntu Cloud Archive 安装和配置 OpenStack 身份认证服务，也就是 Keystone 项目。配置完成之后，连接到 OpenStack 云环境都需经过这里所安装的 OpenStack 身份认证服务。

OpenStack 身份认证服务的默认后台数据库是 MariaDB 数据库。图 1-1 展示我们将要安装的环境。在本章里，我们会专注于 Controller 主机。

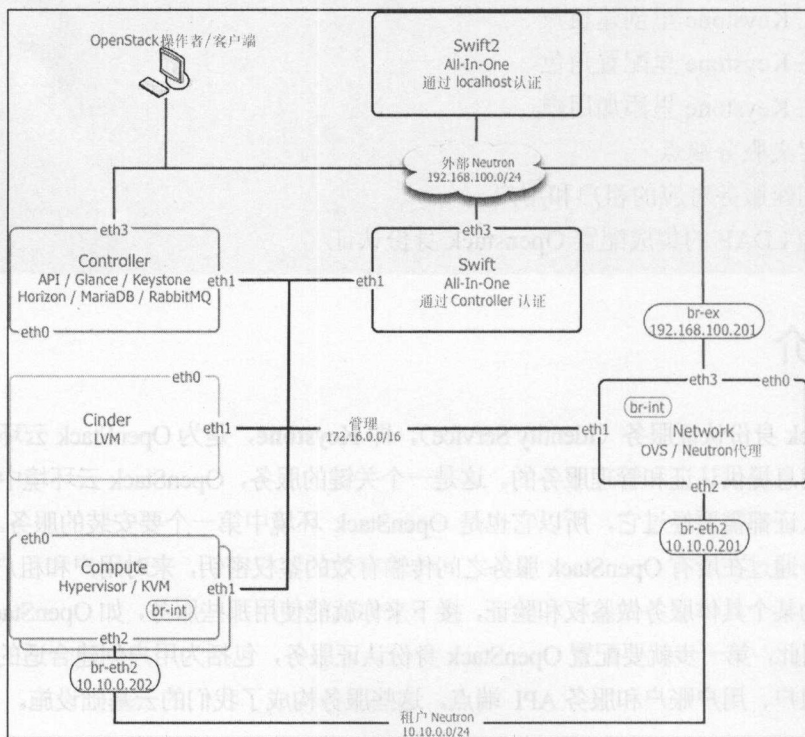


图 1-1

准备工作

为保证运行的是 Ubuntu Cloud Archive，必须先配置 Ubuntu 14.04 安装以使用该服务。更多信息，访问 <http://bit.ly/OpenStackCookbookCloudArchive>。



所有步骤都能在 <http://www.openstackcookbook.com/> 找到。

我们将配置 Keystone，使用 MariaDB 作为数据库后端，因此，安装 Keystone 之前需要安装 MariaDB。



如果 MariaDB 还没有安装，访问 <http://bit.ly/OpenStackCookbookPreReqs>。

请确保你有一台合适的服务器来安装 OpenStack 身份认证服务组件。如果你正在使用前言里描述过的 Vagrant 环境，它就是 controller 节点。

请确保已经登录到 controller 节点上，并且确保能访问因特网，允许我们在环境里安装运行 Keystone 必需的软件包。如果是通过 Vagrant 来创建这个节点，请执行以下命令。

```
vagrant ssh controller
```

上述指令假设 controller 节点有两个 IP 地址。前端 IP 地址 192.168.100.200 和后端 IP 地址 172.16.0.200（它也是 MariaDB 服务器的地址）。存在两个 IP 地址的原因是，内部数据通过后端 IP 地址进行通信（如数据库传输），同时任何 Keystone 的传输都是通过前端 IP 地址。

操作步骤

执行如下指令，安装 OpenStack 身份认证服务。

1. 安装 OpenStack 身份认证服务可通过指定安装 Ubuntu 资源库里的 Keystone 软件包来完成。只需执行如下命令。

```
sudo apt-get update  
sudo apt-get install ntp keystone python-keyring
```

2. 安装好之后，需要配置后台数据库存储。首先需要在 MariaDB 里创建一个 keystone 数据库，按照如下步骤来执行（在本例中，假定 MariaDB 的用户名是 root，对应的密码是 openstack，该用户有创建数据库的权限）。

```
MYSQL_ROOT_PASS=openstack
mysql -uroot -p$MYSQL_ROOT_PASS -e "CREATE DATABASE \
    keystone;"
```

3. 一个最佳实践是在数据库中为 OpenStack 身份认证服务单独创建一个特定的用户。创建命令如下。

```
MYSQL_KEYSTONE_PASS=openstack

mysql -uroot -p$MYSQL_ROOT_PASS -e "GRANT ALL PRIVILEGES ON \
    keystone.* TO 'keystone'@'localhost' IDENTIFIED BY \
    '$MYSQL_KEYSTONE_PASS';"

mysql -uroot -p$MYSQL_ROOT_PASS -e "GRANT ALL PRIVILEGES ON \
    keystone.* TO 'keystone'@'%' IDENTIFIED BY \
    '$MYSQL_KEYSTONE_PASS';"
```

4. 接下来, 配置 OpenStack 身份认证服务来使用该数据库。编辑配置文件/etc/keystone/keystone.conf, 如下所示。

```
[DEFAULT]
admin_token = ADMIN
log_dir=/var/log/keystone

[database]
connection = mysql://keystone:openstack@172.16.0.200/keystone

[extra_headers]
Distribution = Ubuntu
use_syslog = True
syslog_log_facility = LOG_LOCAL0
```

5. 现在重启 keystone 服务来验证这些改动。

```
sudo stop keystone
sudo start keystone
```

6. keystone 启动之后, 用如下命令为 keystone 数据库填充必需的数据表。

```
sudo keystone-manage db_sync
```



恭喜! 现在已经为 OpenStack 环境安装好了 OpenStack 身份认证服务。

工作原理

通过使用 Ubuntu 的包, 可以便捷地安装好 OpenStack 环境中的 OpenStack 身份认证服务。安装完成之后, 在 MariaDB 数据库服务器中配置了 keystone 数据库, 并且用相应值

来设置 `keystone.conf` 配置文件。启动 Keystone 服务之后，运行 `keystone-manage db_sync` 命令来为 keystone 数据库填充合适的数据库表，以方便向其中添加 OpenStack 环境中所必需的用户（user）、角色（role）和租户（tenant）。

1.3 为 SSL 通信配置 OpenStack 身份认证

本书的诸多更新之一是一个更加全面的强化方法。为此，我们开始就默认为 Keystone 服务启用 SSL 通信。请务必注意，我们这里通过自签名的证书来演示如何配置这些服务。在生产中的部署，强烈建议去认证中心（Certificate Authority）获取合适的证书。

准备工作

请确保已经登录 `controller` 节点，并且能访问因特网，允许我们在环境里安装运行 Keystone 必需的软件包。如果你是通过 Vagrant 来创建节点，请执行以下命令。

```
vagrant ssh controller
```

操作步骤

执行如下指令，配置 Keystone 服务。

1. 在配置 Keystone 使用 SSL 通信之前，我们需要生成所需的 OpenSSL 证书。为此，登录正在运行 Keystone 的服务器，执行如下命令。

```
sudo apt-get install python-keystoneclient
keystone-manage ssl_setup --keystone-user keystone \
    --keystone-group keystone
```



生产中，不需要使用 `keystone-manage ssl_setup` 命令。这是一个 Keystone 创建自签名证书的便利工具。

2. 证书生成之后，我们就能用来与 Keystone 服务通信。我们把证书放置在一个可访问的地方，方便在其他服务里引用已生成的 CA 文件。执行以下命令。

```
sudo cp /etc/keystone/ssl/certs/ca.pem /etc/ssl/certs/ca.pem
sudo c_rehash /etc/ssl/certs/ca.pem
```

3. 我们在客户端也使用同样的 CA 和 CA Key 文件，所以把它们复制到将会运行相关 `python-*client` 工具的地方。在 Vagrant 环境里，我们把它复制到主机里，如下所示。

```
sudo cp /etc/keystone/ssl/certs/ca.pem /vagrant/ca.pem
sudo cp /etc/keystone/ssl/certs/cakey.pem /vagrant/cakey.pem
```

4. 然后，编辑 Keystone 配置文件 `/etc/keystone/keystone.conf`，增加如下内容。


```
[ssl]
enable = True
certfile = /etc/keystone/ssl/certs/keystone.pem
keyfile = /etc/keystone/ssl/private/keystonekey.pem
ca_certs = /etc/keystone/ssl/certs/ca.pem
cert_subject=/C=US/ST=Unset/L=Unset/O=Unset/CN=192.168.100.200
ca_key = /etc/keystone/ssl/certs/cakey.pem
```

5. 最后，重启 Keystone 服务。

```
sudo stop keystone
sudo start keystone
```

工作原理

Openstack 服务一般通过标准 HTTP 请求来完成互通。这提供了很大程度的灵活性，但这是以所有的通信都是基于明文为代价的。通过增加 SSL 认证，并且修改 Keystone 的配置，所有 Keystone 的通信都将通过 HTTPS 加密。

1.4 在 Keystone 里创建租户

一个租户 (tenant) 在 OpenStack 里就是一个项目，这两个术语通常互换使用。在创建一个用户时必须首先为该用户分配一个租户，否则将无法创建此用户，所以首先要创建租户。在这一节中，将为用户创建一个名为 cookbook 的租户。

准备工作

我们将会使用 keystone 客户端来操作 Keystone。如果工具 python-keystoneclient 不可用，请按照如下所描述的步骤操作：<http://bit.ly/OpenStackCookbookClientInstall>。

为了能以管理者权限访问 OpenStack 环境，请确保已经正确设置环境。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```



如果网络中没有其他可用的机器，可以使用 controller 节点，因为它有 python-keystoneclient，并且可以访问 OpenStack 环境。如果使用 Vagrant 环境，请执行以下命令来访问 Controller:

```
vagrant ssh controller
```

操作步骤

执行如下步骤，在 OpenStack 环境中创建一个租户。

1. 我们从创建一个名为 `cookbook` 的租户开始。

```
keystone tenant-create\
  --name cookbook\
  --description "Default Cookbook Tenant"\
  --enabled true
```

输出如图 1-2 所示。

Property	Value
description	Default Cookbook Tenant
enabled	True
id	fba7b31689714d1ab39a751bc9483efd
name	cookbook

图 1-2

2. 同样需要一个 `admin` 租户，该租户下的用户能够访问整个环境。因此，使用同样的方式。

```
keystone tenant-create\
  --name admin \
  --description "Admin Tenant" \
  --enabled true
```

工作原理

通过 keystone 客户端很容易创建租户，只需要指定 `tenant-create` 相关选项，语法如下所示。

```
keystone tenant-create \
  --name tenant_name \
  --description "A description" \
  --enabled true
```

`tenant_name` 是一个不包含空格的任意字符串。创建租户时，keystone 会返回一个与租户相对应的 ID，可以通过这个 ID 来将用户添加到这个租户里。如果想查看环境中所有租户和它们所对应 ID 的列表，可以执行如下命令。

```
keystone tenant-list
```

1.5 在 Keystone 里配置角色

角色是分配给一个租户中用户的权限。在这里配置两个角色，一个用于管理云环境的 admin 角色和另一个用于分配给使用云环境的普通用户的 member 角色。

准备工作

我们将会使用 keystone 客户端来操作 Keystone。如果工具 python-keystoneclient 不可用，请按照如下所描述的步骤操作：<http://bit.ly/OpenStackCookbookClientInstall>。

为了能以管理者权限访问 OpenStack 环境，请确保已经正确设置环境。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```



如果网络中没有其他可用的机器，可以使用 controller 节点，因为它有 python-keystoneclient，并且可以访问 OpenStack 环境。如果使用 Vagrant 环境，执行以下命令来访问 Controller:

```
vagrant ssh controller
```

操作步骤

执行如下步骤，在我们的 OpenStack 环境中创建必需的角色。

1. 创建 admin 角色。

```
# admin role
keystone role-create --name admin
```

输出如图 1-3 所示。

Property	Value
id	625b81ae9f024366bbe023a62ab8a18d
name	admin

图 1-3

2. 用同样的命令来创建 Member role，只是把 name 选项指定为 Member。

```
# Member role
keystone role-create --name Member
```

工作原理

通过 keystone 客户端创建一个角色很容易实现，只需要在运行 keystone 时指定 role-create 选项，语法如下所示。

```
keystone role-create --name role_name
```

对 admin 和 Member role 来说，role_name 的属性值不是任意的。admin 角色已经在 /etc/keystone/policy.json 配置文件中被设置为具有管理员权限。

```
{
  "admin_required": [["role:admin"], ["is_admin:1"]]
}
```

通过 Web 接口创建的非管理员用户，会在 OpenStack Dashboard（即 Horizon）中默认配置成 Member role。

创建角色时，keystone 会返回一个与角色相对应的 ID，可以通过这个 ID 来把角色关联到某个用户。如果想查看环境中所有角色和它们所对应 ID 的列表，可以执行如下命令。

```
keystone role-list
```

1.6 往 Keystone 里添加用户

在 OpenStack 身份认证服务中添加用户时，必须要有一个能容纳该用户的租户，还需要定义一个能分配给该用户的角色。在本节中，将创建两个用户。第一个用户名为 admin，它在 cookbook 租户中被分配为 admin 角色。第二个用户名为 demo，同样在 cookbook 租户中，它被分配为 Member 角色。

准备工作

我们将会使用 keystone 客户端来操作 Keystone。如果工具 python-keystoneclient 不可用，请按照如下所描述的步骤操作：<http://bit.ly/OpenStackCookbookClientInstall>。

为了能以管理者权限访问 OpenStack 环境，请确保已经正确设置环境。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
```



```
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/keystonerc
export OS_CACERT=/vagrant/ca.pem
```



如果网络中没有其他可用的机器,可以使用 controller 节点,因为它有 python-keystoneclient, 并且可以访问 OpenStack 环境。如果使用 Vagrant 环境,执行以下命令来访问 Controller:

```
vagrant ssh controller
```

操作步骤

执行如下步骤,在 OpenStack 环境中创建用户。

1. 如果要在 cookbook 租户中创建一个用户,首先要获得 cookbook 租户的 ID。通过 keystone 命令,指定 tenant-list 选项,就可以得到该 ID,然后将其存储在 TENANT_ID 变量中,命令如下所示。

```
TENANT_ID=$(keystone tenant-list \
| awk '/\ cookbook\ / {print $2}')
```

2. 现在已经得到了租户的 ID,在 cookbook 租户中创建 admin 用户,注意要使用 user-create 选项,还需要为该用户设置密码。

```
PASSWORD=openstack
keystone user-create \
--name admin \
--tenant_id $TENANT_ID \
--pass $PASSWORD \
--email root@localhost \
--enabled true
```

输出如图 1-4 所示。

Property	Value
email	root@localhost
enabled	True
id	2e23d0673e8a4deabe7c0fb70dfcb9f2
name	admin
tenantId	14e34722ac7b4fe298886371ec17cf40
username	admin

图 1-4

3. 在创建 admin 用户时，为了赋予它 admin 角色，需要先获得 admin 角色的 ID。用 role-list 选项取出 admin 角色的 ID，然后将其存储在一个变量里。

```
ROLE_ID=$(keystone role-list \
| awk '/\ admin\ / {print $2}')
```

4. 为了向角色赋予 admin 用户，需要用到创建 admin 用户时返回的用户 ID。执行如下的 keystone 命令，通过 user-list 选项列出所有的用户，从而得到 admin 用户的 ID。

```
USER_ID=$(keystone user-list \
| awk '/\ admin\ / {print $2}')
```

5. 根据租户 ID、用户 ID 以及对应的角色 ID，通过 user-role-add 选项把角色赋予对应的用户。

```
keystone user-role-add \
--user $USER_ID \
--role $ROLE_ID \
--tenant_id $TENANT_ID
```



注意，成功执行该命令之后是没有输出的。

6. 为了管理整个环境，admin 用户也需要在 admin 租户中。为此，需要获得 admin 租户的 ID 并使用新租户的 ID 重复前面的步骤。

```
ADMIN_TENANT_ID=$(keystone tenant-list \
| awk '/\ admin\ / {print $2}')
keystone user-role-add \
--user $USER_ID \
--role $ROLE_ID \
--tenant_id $ADMIN_TENANT_ID
```

7. 接下来要在 cookbook 租户里创建一个 demo 用户，并赋予其 Member 角色，操作类似前 5 步，命令如下所示。

```
# Get the cookbook tenant ID
TENANT_ID=$(keystone tenant-list \
| awk '/\ cookbook\ / {print $2}')

# Create the user
PASSWORD=openstack
keystone user-create \
--name demo \
--tenant_id $TENANT_ID \
--pass $PASSWORD \
--email demo@localhost \
--enabled true
```

```
# Get the Member role ID
ROLE_ID=$(keystone role-list \
| awk '/\ Member\ / {print $2}')

# Get the demo user ID
USER_ID=$(keystone user-list \
| awk '/\ demo\ / {print $2}')

# Assign the Member role to the demo user in cookbook
keystone user-role-add \
--user $USER_ID \
--role $ROLE_ID \
--tenant_id $TENANT_ID
```

工作原理

向 OpenStack 身份认证服务里添加用户涉及多个步骤和依赖关系。首先，需要有租户，才能容纳用户。只要租户存在，就可以增加用户。这时，用户还没有关联的角色，所以最后一步就是给用户指定角色，比如 Member 或 admin。

通过 user-create 选项，使用如下语法创建用户。

```
keystone user-create \
--name user_name \
--tenant_id TENANT_ID \
--pass PASSWORD \
--email email_address \
--enabled true
```

user_name 属性可以是任意名称，但不能包含空格。password 属性是必需的，在之前的例子里，它们都被设为 openstack。email_address 属性也是必需的。

赋予一个用户某个角色的命令选项是 user-role-add，语法如下所示。

```
keystone user-role-add \
--user USER_ID \
--role ROLE_ID \
--tenant_id TENANT_ID
```

这表示在赋予角色之前，必须先取得用户的 ID、角色的 ID 及租户的 ID。这些 ID 可以通过如下命令得到。

```
keystone tenant-list
keystone user-list
keystone role-list
```

1.7 定义服务端点

云环境中的每一个服务都运行在一个特定的 URL 和端口上,也就是这些服务的端点地址。当一个客户端程序连到 OpenStack 环境时,Keystone 身份认证服务负责向其返回云环境中的各个服务的端点地址,以便客户端程序使用这些服务。为启用该功能,必须先定义这些端点。在云环境中,可以定义多个区域,可以把不同的区域理解为不同的数据中心,它们各自有不同的 URL 或 IP 地址。在 OpenStack 身份认证服务里,还可以在每一个区域里分别定义 URL 端点。在这里,由于只有一个区域,将其标识为 RegionOne。

准备工作

我们将会使用 keystone 客户端来操作 Keystone。如果工具 python-keystoneclient 不可用,请按照如下所描述的步骤操作:<http://bit.ly/OpenStackCookbookClientInstall>。

为了能以管理者权限访问 OpenStack 环境,请确保已经正确设置环境。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```



如果网络中没有其他可用的机器,可以使用 controller 节点,因为它有 python-keystoneclient,并且可以访问 OpenStack 环境。

如果使用 Vagrant 环境,请执行以下命令来访问 Controller:

```
vagrant ssh controller
```

操作步骤

通过运行 keystone 客户端命令,可在 OpenStack 身份认证服务中定义各个服务和服务端点。即使某些服务现在还没有在云环境中运行起来,也可以先在 OpenStack 身份认证服务中配置好,以便将来使用。通过如下步骤,在 OpenStack 环境中为各个服务定义端点。

1. 现在来定义一些云环境中 OpenStack 身份认证服务需要知道的服务。

```
# OpenStack Compute Nova API Endpoint
keystone service-create \
  --name nova \
```



```

--type compute \
--description 'OpenStack Compute Service'
# OpenStack Compute EC2 API Endpoint
keystone service-create \
--name ec2 \
--type ec2 \
--description 'EC2 Service'

# Glance Image Service Endpoint
keystone service-create \
--name glance \
--type image \
--description 'OpenStack Image Service'

# Keystone Identity Service Endpoint
keystone service-create \
--name keystone \
--type identity \
--description 'OpenStack Identity Service'

# Neutron Networking Service Endpoint
keystone service-create \
--name network \
--type network \
--description 'OpenStack Network Service'

# Cinder Block Storage Endpoint
keystone service-create \
--name volume \
--type volume \
--description 'Volume Service'

```

2. 上述操作完成之后，可以在服务里增加这些服务运行的端点 URL。添加服务端点 URL 时需要用到各个服务的 ID，这些 ID 在上一步命令操作之后会分别被返回输出。它们是配置服务端口 URL 的命令中的参数。



OpenStack 身份认证服务可以被配置成在三个 URL 上接受服务请求：一个面向公众的 URL（被终端用户使用），一个面向管理员的 URL（被以管理员权限登录的用户使用，可以是一个不同的 URL），以及一个面向内部的 URL（当这些服务是在公有的 URL 的防火墙内提供服务时）。

3. 对于如下服务，我们将会配置独立的公有、管理员和内部服务 URL，为我们的环境提供适当的隔离。实验室环境里的公有端点，将会被指定公共 IP，192.168.100.200。内部端点是 172.16.0.200。管理员端点也是一个公共 IP，192.168.100.200。通过如下命令来完成。

```
# OpenStack Compute Nova API
```

```
NOVA_SERVICE_ID=$(keystone service-list \
| awk '/\ nova\ / {print $2}')
```

```
PUBLIC_ENDPOINT=192.168.100.200
ADMIN_ENDPOINT=192.168.100.200
INT_ENDPOINT=172.16.0.200
```

```
PUBLIC="http://$PUBLIC_ENDPOINT:8774/v2/\$(tenant_id)s"
ADMIN="http://$ADMIN_ENDPOINT:8774/v2/\$(tenant_id)s"
INTERNAL="http://$INT_ENDPOINT:8774/v2/\$(tenant_id)s"
```

```
keystone endpoint-create \
--region RegionOne \
--service_id $NOVA_SERVICE_ID \
--publicurl $PUBLIC \
--adminurl $ADMIN \
--internalurl $INTERNAL
```

输出如图 1-5 所示。

Property	Value
adminurl	http://192.168.100.200:8774/v2/\\$(tenant_id)s
id	87b59c5ce8314d8b9029b1efd5044d7
internalurl	http://172.16.0.100:8774/v2/\\$(tenant_id)s
publicurl	http://192.168.100.200:8774/v2/\\$(tenant_id)s
region	RegionOne
service_id	a3529dcbeab44d479d1f258ae6d202b4

图 1-5

4. 继续定义其他服务端点，具体步骤如下。

```
# OpenStack Compute EC2 API
EC2_SERVICE_ID=$(keystone service-list \
| awk '/\ ec2\ / {print $2}')
```

```
PUBLIC="http://$PUBLIC_ENDPOINT:8773/services/Cloud"
ADMIN="http://$ADMIN_ENDPOINT:8773/services/Admin"
INTERNAL="http://$INT_ENDPOINT:8773/services/Cloud"
```

```
keystone endpoint-create \
--region RegionOne \
--service_id $EC2_SERVICE_ID \
--publicurl $PUBLIC \
--adminurl $ADMIN \
--internalurl $INTERNAL
```

```
# Glance Image Service
GLANCE_SERVICE_ID=$(keystone service-list \
| awk '/\ glance\ / {print $2}')
```

```
PUBLIC="http://$PUBLIC_ENDPOINT:9292/v1"
```

```
ADMIN="http://$ADMIN_ENDPOINT:9292/v1"
INTERNAL="http://$INT_ENDPOINT:9292/v1"
```

```
keystone endpoint-create \
  --region RegionOne \
  --service_id $GLANCE_SERVICE_ID \
  --publicurl $PUBLIC \
  --adminurl $ADMIN \
  --internalurl $INTERNAL
```

```
# Keystone OpenStack Identity Service
```

```
# Note we're using SSL HTTPS here
```

```
KEYSTONE_SERVICE_ID=$(keystone service-list \
  | awk '/\ keystone\ / {print $2}')
```

```
PUBLIC="https://$PUBLIC_ENDPOINT:5000/v2.0"
ADMIN="https://$ADMIN_ENDPOINT:35357/v2.0"
INTERNAL="https://$INT_ENDPOINT:5000/v2.0"
```

```
keystone endpoint-create \
  --region RegionOne \
  --service_id $KEYSTONE_SERVICE_ID \
  --publicurl $PUBLIC \
  --adminurl $ADMIN \
  --internalurl $INTERNAL
```

```
# Neutron Networking Service
```

```
NEUTRON_SERVICE_ID=$(keystone service-list \
  | awk '/\ network\ / {print $2}')
```

```
PUBLIC="http://$PUBLIC_ENDPOINT:9696"
```

```
ADMIN="http://$ADMIN_ENDPOINT:9696"
```

```
INTERNAL="http://$INT_ENDPOINT:9696"
```

```
keystone endpoint-create \
  --region RegionOne \
  --service_id $NEUTRON_SERVICE_ID \
  --publicurl $PUBLIC \
  --adminurl $ADMIN \
  --internalurl $INTERNAL
```

```
# Cinder Block Storage Service
```

```
CINDER_SERVICE_ID=$(keystone service-list \
  | awk '/\ volume\ / {print $2}')
```

```
PUBLIC="http://$PUBLIC_ENDPOINT:8776/v1/%(tenant_id)s"
```

```
ADMIN=$PUBLIC
```

```
INTERNAL=$PUBLIC
```

```
keystone endpoint-create \
  --region RegionOne \
  --service_id $CINDER_SERVICE_ID \
  --publicurl $PUBLIC \
```



```
--adminurl $ADMIN \
--internalurl $INTERNAL
```

工作原理

在 OpenStack 身份认证服务中配置服务和端点是通过 keystone 客户端命令来完成的。首先通过 keystone 客户端的 service-create 选项来添加服务，语法如下所示。

```
keystone service-create \
--name service_name \
--type service_type \
--description 'description'
```

service_name 可以是任意的名字或标签，用于标识服务的类型。在定义端点的时候，可以通过这个名字来取得它对应的 ID。

type 选项可以是下列几个选项中的一个：compute、object-store、image-service、network 和 identity-service。注意，在这一步没有配置 OpenStack 的对象存储服务（type 是 object-store），这会在本书后续章节涉及。

description 字段同样是一个任意字段，用于描述该服务。

添加好各个服务之后，通过 keystone 客户端的 endpoint-create 选项来定义各个服务对应的端点。只有这样，OpenStack 身份认证服务才能知道如何访问它们。语法如下所示。

```
keystone endpoint-create \
--region region_name \
--service_id service_id \
--publicurl public_url \
--adminurl admin_url \
--internalurl internal_url
```

service_id 是在第一步里创建的服务的 ID。可以通过如下命令来列出所有服务和它们对应的 ID。

```
keystone service-list
```

OpenStack 被设计为一个适合全球部署的系统，一个区域（region）表示一个实际的数据中心或者一个包含多个互相连通的数据中心的地域范围。为此，我们只定义了一个区域——RegionOne。region 字段可以是任意的名字，用于标识数据中心/地域。在指定哪个数据中心/地域运行哪些服务时，可引用该名称，也可用来告知客户端使用哪些区域。

所有的服务都可以被配置成运行在 3 个不同 URL 之上，如下所述，这取决于人们希望如何来配置 OpenStack 云环境。

- ❑ `public_url`: 是供终端用户连接的 URL。在一个公有云环境中, 这将是一个公有的 URL, 并解析成公有的 IP 地址。
- ❑ `admin_url`: 只用于管理员访问。在公有部署中, 通常会配置一个和 `public_url` 不同的 `admin_url`。有些模块的管理服务有不同的 URL, 这就需要配置这个属性。
- ❑ `internal_url`: 只用于本地私有网络。该参数将是私有本地网络中才存在的 IP 或 URL。通过 `internal_url`, 你可以从云环境内部连接到各个服务, 而不需要通过公共 IP 地址空间, 因此避免了因特网访问流量费用。这也带来了更好的安全性和更低的复杂性。



首先创建好初始化的 keystone 数据库, 然后在 OpenStack 身份认证服务器上运行 `keystone-manage db_sync` 命令, 最后就可以使用 keystone 客户端来实现远程管理了。

1.8 创建服务的租户和服务的用户

服务端点创建完成以后, 接下来配置它们以便其他 OpenStack 服务能够调用。为此, 要为每个服务都配置一个特定的 `service` 租户, 并指定对应的用户名和密码。这样的目的是保证更高的安全性, 以及用来为云环境做故障排除和审计等工作。当设置一个服务使用 OpenStack 身份认证服务来验证和授权时, 我们都会在它们相关的配置文件里指明这些细节。为了在 OpenStack 里可用, 每一个服务都需要经过 keystone 认证。通过这些证书可以完成服务的配置。例如, 当通过 OpenStack 身份认证服务使用 `glance` 服务时, 要在 `/etc/glance/glance-registry.conf` 文件中指定如下这些配置, 且必须与之前创建的相匹配。

```
[keystone_authtoken]
identity_uri = https://192.168.100.200:35357
admin_tenant_name = service
admin_user = glance
admin_password = glance
insecure = True
```



在本书里, `insecure = True` 只是自签名的证书才需要。在生产中, 我们应该使用颁发的证书, 并且在配置里忽略这个选项。

准备工作

我们将会使用 keystone 客户端来操作 Keystone。如果工具 `python-keystoneclient` 不可用, 请按照如下所描述的步骤操作: <http://bit.ly/OpenStackCookbookClientInstall>。

为了能以管理者权限访问 OpenStack 环境，请确保已经正确设置环境。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/akey.pem
export OS_CACERT=/vagrant/ca.pem
```



如果网络中没有其他可用的机器，可以使用 controller 节点，因为它有 python-keystoneclient，并且可以访问 OpenStack 环境。如果使用 Vagrant 环境，执行以下命令来访问 Controller:

```
vagrant ssh controller
```

操作步骤

执行如下步骤，配置一个合适的服务租户。

1. 创建服务租户，命令如下所示。

```
keystone tenant-create \
  --name service \
  --description "Service Tenant" \
  --enabled true
```

输出如图 1-6 所示。

Property	Value
description	Service Tenant
enabled	True
id	8e77d9c13e884bf4809077722003bba0
name	service

图 1-6

2. 记录下服务租户的 ID，以方便后面指定服务用户时使用。

```
SERVICE_TENANT_ID=$(keystone tenant-list \
  | awk '/\ service\ / {print $2}')
```

3. 对于本节中的所有服务，都会为其创建一个用户账户，用户名和密码都和服务名称一致。例如，会在 service 租户里用 user-create 选项创建一个用户名和密码均为 nova 的用户，命令如下所示。


```
keystone user-create \
  --name nova \
  --pass nova \
  --tenant_id $SERVICE_TENANT_ID \
  --email nova@localhost \
  --enabled true
```

上述代码输出如图 1-7 所示。

Property	Value
email	nova@localhost
enabled	True
id	50ea356a4b6f4cb7a9fa22c1fb08549b
name	nova
tenantId	42e5c284de244e3190e12cc44fbbbe62
username	nova

图 1-7

4. 重复上一步，继续为其他用到 OpenStack 身份认证功能的服务创建用户。

```
keystone user-create \
  --name glance \
  --pass glance \
  --tenant_id $SERVICE_TENANT_ID \
  --email glance@localhost \
  --enabled true
```

```
keystone user-create \
  --name keystone \
  --pass keystone \
  --tenant_id $SERVICE_TENANT_ID \
  --email keystone@localhost \
  --enabled true
```

```
keystone user-create \
  --name neutron \
  --pass neutron \
  --tenant_id $SERVICE_TENANT_ID \
  --email neutron@localhost \
  --enabled true
```

```
keystone user-create \
  --name cinder \
  --pass cinder \
  --tenant_id $SERVICE_TENANT_ID \
  --email cinder@localhost \
  --enabled true
```

5. 现在为这些服务租户里的用户分配 admin 角色。为此，需要先取得 nova 用户的用户 ID，然后再用 user-role-add 选项来分配角色。例如，为了将 admin 角色分配给服务租户里的 nova 用户，使用如下命令。

```
# Get the nova user id
NOVA_USER_ID=$(keystone user-list \
| awk '/\ nova\ / {print $2}')
```

```
# Get the admin role id
ADMIN_ROLE_ID=$(keystone role-list\
| awk '/\ admin\ / {print $2}')
```

```
# Assign the nova user the admin role in service tenant
keystone user-role-add\
--user $NOVA_USER_ID\
--role $ADMIN_ROLE_ID\
--tenant_id $SERVICE_TENANT_ID
```

6. 接下来，重复上一步，继续为其他服务租户 (glance、keystone、neutron 和 cinder 用户) 分配角色。

```
# Get the glance user id
GLANCE_USER_ID=$(keystone user-list\
| awk '/\ glance\ / {print $2}')
```

```
# Assign the glance user the admin role in service tenant
keystone user-role-add\
--user $GLANCE_USER_ID\
--role $ADMIN_ROLE_ID\
--tenant_id $SERVICE_TENANT_ID
```

```
# Get the keystone user id
KEYSTONE_USER_ID=$(keystone user-list\
| awk '/\ keystone\ / {print $2}')
```

```
# Assign the keystone user the admin role in service tenant
keystone user-role-add\
--user $KEYSTONE_USER_ID\
--role $ADMIN_ROLE_ID\
--tenant_id $SERVICE_TENANT_ID
```

```
# Get the cinder user id
NEUTRON_USER_ID=$(keystone user-list \
| awk '/\ neutron \ / {print $2}')
```

```
# Assign the neutron user the admin role in service tenant
keystone user-role-add \
--user $NEUTRON_USER_ID \
--role $ADMIN_ROLE_ID \
--tenant_id $SERVICE_TENANT_ID
```

```
# Get the cinder user id
CINDER_USER_ID=$(keystone user-list\
| awk '/\ cinder \ / {print $2}')

# Assign the cinder user the admin role in service tenant
keystone user-role-add\
--user $CINDER_USER_ID\
--role $ADMIN_ROLE_ID\
--tenant_id $SERVICE_TENANT_ID
```

工作原理

创建服务租户，然后在其中添加 OpenStack 运行必需的各个服务，这两步操作和在系统中添加需要 admin 角色的其他用户没什么不同。为每个拥有 admin 角色的用户分配用户名和密码，并确保它们存在于服务租户内。然后，使用这些凭证来配置服务与 OpenStack 身份认证服务进行验证。



本书示例代码可以从以下网址下载：<https://github.com/OpenStackCookbook/OpenStackCookbook>。所有的支持文件都在这里。

1.9 为 LDAP 的集成配置 Openstack 身份认证

迄今为止我们构建的 OpenStack 身份认证服务，为 OpenStack 环境提供了一个功能性但孤立的设置。这是一个有用的概念验证和实验室环境。但是，我们很可能需要把 OpenStack 集成到已经存在的认证系统里。OpenStack 为这种场景提供了一种可插拔的认证后端，其中使用最广泛的是 LDAP。

准备工作

我们将会使用 keystone 客户端来操作 Keystone。如果工具 python-keystoneclient 不可用，请按照如下所描述的步骤操作：<http://bit.ly/OpenStackCookbookClientInstall>。

为了能以管理者权限访问 OpenStack 环境，请确保已经正确设置环境。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```




如果网络中没有其他可用的机器，可以使用 controller 节点，因为它有 python-keystoneclient，并且可以访问 OpenStack 环境。如果使用 Vagrant 环境，执行以下命令来访问 Controller:

```
vagrant ssh controller
```

另外，如果要连接到一个外部 LDAP 服务，你需要持有 LDAP 的主机名或 IP 地址，并且有合适的访问权限。你需要拥有管理者用户的 LDAP 路径信息，和包含用户、角色和租户的组织单位。



作为本书的补充材料，我们提供了一个示例 OpenLDAP 服务器，预置了所需的值。关于如何使用的步骤，参考本书的博客：<http://bit.ly/OpenStackCookbookLDAP>。

操作步骤

为了与 LDAP 通信，我们通过如下步骤配置 OpenStack 身份认证服务。

1. 使用你最喜欢的编辑器，在 keystone.conf 文件中启用 LDAP 验证。

```
[identity]
driver=keystone.identity.backends.ldap.Identity
```

2. 接下来，创建 ldap 字段，添加 LDAP 服务器 URL。

```
[ldap]
url = ldap://openldap
```

3. 在如下行里，为你使用的管理员用户指定 LDAP 路径，同时指定密码和后缀，或者

Keystone 开始搜索 LDAP 的地址。

```
user = cn=admin,dc=cook,dc=book
password = openstack
suffix = cn=cook,cn=book
```

4. 仍然在 [ldap] 字段，我们告知 Keystone 关于如何查找用户的 4 个信息。user_tree_dn 指定在 LDAP 树里为用户搜索哪个 OU。user_objectclass 指定 LDAP 中如何表示用户。user_id_attribute 告知 Keystone 哪个属性会被用作用户的用户名。类似地，user_mail_attribute 告知 Keystone 去哪儿查找用户的电子邮件地址，代码如下。

```
user_tree_dn = ou=Users,dc=cook,dc=book
user_objectclass = inetOrgPerson
user_id_attribute = cn
user_mail_attribute = mail
```

5. 接下来，为租户和角色增加相同细节。

```
tenant_tree_dn = ou=Projects,dc=cook,dc=book
tenant_objectclass = groupOfNames
```

```

tenant_id_attribute = cn
tenant_desc_attribute = description

role_tree_dn = ou=Roles,dc=cook,dc=book
role_objectclass = organizationalRole
role_id_attribute = cn
role_member_attribute = roleOccupant

```

6. 保存文件，重启 keystone。

```

sudo stop keystone
sudo start keystone

```

工作原理

OpenStack 身份认证服务，就像其他 OpenStack 服务一样，是基于插件的。在默认状态下，Keystone 从 SQL 数据库里存储和访问所有的用户身份和认证数据。但是，当把 OpenStack 集成到一个已经存在的环境时，这就并非总是最可取或安全的方法了。为了适应这点，我们把认证后端改为 LDAP。这样，我们可以集成 OpenLDAP、Active Directory 和其他很多服务。但是，在配置后端时，需要特别关注 LDAP 的路径。



服务目录的入口在哪里？它们仍然被存储在 Keystone 的 SQL 数据库里，因为与用户的身份或认证无关。

第 2 章

Glance——OpenStack 镜像服务

本章将讲述以下内容：

- ☐ 安装 OpenStack 镜像服务
- ☐ 用 OpenStack 身份认证服务配置 OpenStack 镜像服务
- ☐ 用 OpenStack 对象存储配置 OpenStack 镜像服务
- ☐ 用 OpenStack 镜像服务管理镜像
- ☐ 注册一个远程存储的镜像
- ☐ 在租户间共享镜像
- ☐ 查看共享镜像
- ☐ 使用镜像元数据
- ☐ 迁移 VMware 镜像
- ☐ 创建一个 OpenStack 镜像

2.1 简介

OpenStack 镜像服务 (Image Service)，也称为 Glance，可以让用户注册、查找和检索在 OpenStack 环境中使用的虚拟机镜像。OpenStack 镜像服务支持将镜像文件存储在各种类型的存储环境，例如本地文件系统或分布式文件系统，如 OpenStack 对象存储服务。

在本章中，我们重点关注如图 2-1 所示的 Controller 主机。

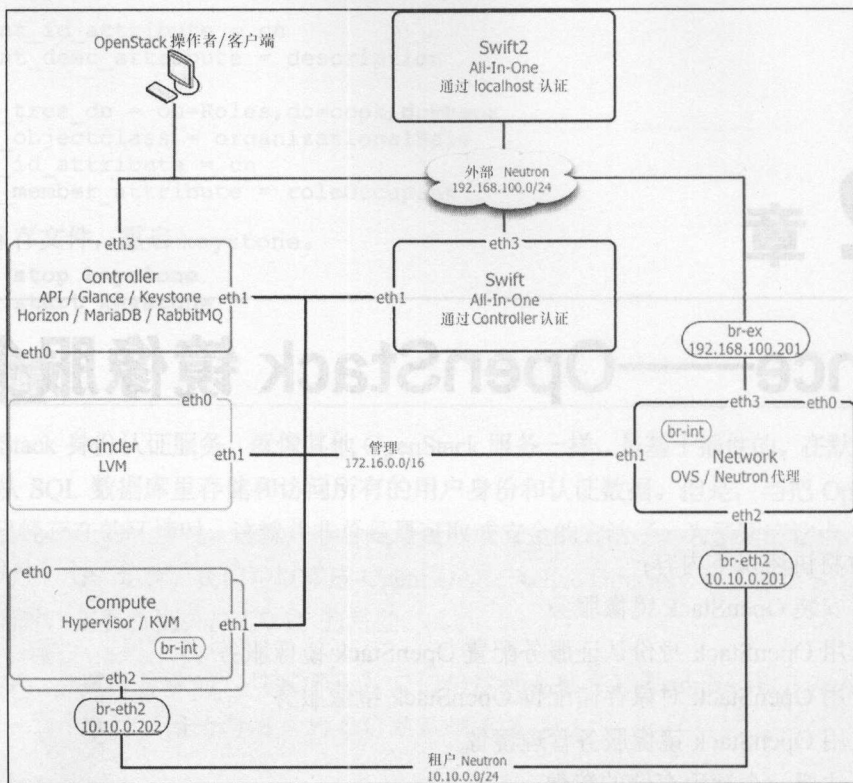


图 2-1

2.2 安装 OpenStack 镜像服务

最新 OpenStack 镜像服务的安装很简单，只需要使用 Ubuntu Cloud Archive 资源库中专门为 Ubuntu 14.04 版本打包好的安装包即可。

准备工作

在开始之前，必须确认已经登录到安装好 OpenStack 身份认证服务的 OpenStack 控制节点上。

执行以下命令，登录到使用 Vagrant 创建的 OpenStack 控制节点。

vagrant ssh controller

确保 Ubuntu 14.04 LTS 版本正在使用 Ubuntu Cloud Archive, 其中包含有 Juno 版本所需的安装包。更多信息请访问: <http://bit.ly/OpenStackCookbookCloudArchive>。



所有步骤都能在如下网站找到: <http://www.openstackcookbook.com/>。

我们使用 MariaDB 作为数据库后端来配置 Glance, 所以它需要在 Glance 之前安装。



如果 MariaDB 没有安装, 访问 <http://bit.ly/OpenStackCookbookInstallMariaDB>。

我们也需要安装 RabbitMQ 作为消息队列服务, 所以它也需要在 Glance 之前安装。



如果 RabbitMQ 没有安装, 访问 <http://bit.ly/OpenStackCookbookInstallRabbitMQ>。

本小节的指令假设 controller 节点有两个 IP 地址。前端 IP 地址 192.168.100.200 和后端 IP 地址 172.16.0.200 (它也是 MariaDB 服务器的地址)。存在两个 IP 地址的原因是, 内部数据通过后端 IP 地址进行通信 (如数据库传输), 任何 Glance 的传输都是通过前端 IP 地址。

操作步骤

通过如下步骤安装 OpenStack 镜像服务。

1. 通过在 Ubuntu 里指定 glance 软件包来完成 OpenStack 镜像服务的安装。

```
sudo apt-get update
sudo apt-get install ntp glance python-keyring
```

2. 安装完成之后, 需要配置后端数据库。我们首先在 MariaDB 里创建 glance 数据库。

```
MYSQL_ROOT_PASS=openstack
mysql -uroot -p$MYSQL_ROOT_PASS -e "CREATE DATABASE \
glance;"
```

在 MariaDB 里有一个名为 root 的用户, 密码是 openstack, 拥有创建数据库的权限。

3. 一个好的实践是创建一个 OpenStack 镜像服务特有的用户, 所以我们在数据库里创建一个 glance 用户。

```
MYSQL_GLANCE_PASS=openstack
```

```
mysql -uroot -p$MYSQL_ROOT_PASS -e "GRANT ALL PRIVILEGES ON \
glance.* TO 'glance'@'localhost' IDENTIFIED BY \
'$MYSQL_KEYSTONE_PASS';"
```

```
mysql -uroot -p$MYSQL_ROOT_PASS -e "GRANT ALL PRIVILEGES ON \
glance.* TO 'glance'@'%' IDENTIFIED BY '$MYSQL_GLANCE_PASS';"
```

4. 现在用这个数据库来配置 OpenStack 镜像服务, 编辑/etc/glance/glance-registry.

conf 和 /etc/glance/glance-api.conf 文件，修改 `sql_connection` 这一行，使它与数据库的证书相匹配。在文件里修改如下行。

```
[database]
backend = sqlalchemy
connection = mysql://glance:openstack@172.16.0.200/glance
```

5. 配置 OpenStack 镜像服务来使用 RabbitMQ，确保 /etc/glance/glance-registry.

conf 中有如下行。

```
rabbit_host = localhost
rabbit_port = 5672
rabbit_use_ssl = false
rabbit_userid = guest
rabbit_password = guest
rabbit_virtual_host = /
rabbit_notification_exchange = glance
rabbit_notification_topic = notifications
rabbit_durable_queues = False
```

6. 现在重启 glance-registry 服务。

```
sudo stop glance-registry
sudo start glance-registry
```

7. 重启 glance-api 服务。

```
sudo stop glance-api
sudo start glance-api
```

8. glance 数据库在 Ubuntu 14.04 下是有版本控制的，允许服务的升级和降级。我们通过如下命令把版本控制设置为 0。

```
sudo glance-manage db_version_control 0
```

9. 我们现在同步一下数据库，确保存在正确的表结构，命令如下。

```
sudo glance-manage db_sync
```

恭喜你！现在 OpenStack 镜像服务已经安装成功，并且可以在我们的 OpenStack 环境中使用了。

工作原理

OpenStack 镜像服务被分为两个运行的服务：glance-api 和 glance-registry。glance-registry 服务连接数据库后端。第一步是创建 glance 数据库和 glance 用户，接下来就可以操作我们创建的 glance 数据库了。

数据库创建后，我们修改 /etc/glance/glance-registry.conf 和 /etc/glance/glance-api.conf 文件，这样 glance 知道去哪儿查找和连接我们的 MySQL 数据库。标准的 SQLAlchemy 连接字符串语法如下。

```
sql_connection = mysql://USER:PASSWORD@HOST/DBNAME
```


延伸阅读

□ 参见第 1 章。

2.3 用 OpenStack 身份认证服务配置 OpenStack 镜像服务

为了保证 OpenStack 计算服务正确运行，必须对 OpenStack 镜像服务作适当配置，以保证其能正常使用 OpenStack 身份认证服务。

准备工作

在开始之前，确保已经登录到安装了 OpenStack 身份认证服务的 OpenStack 控制主机上。如果 OpenStack 身份认证服务没有安装，执行第 1 章 1.2 节中的步骤。我们也需要设置好 Glance 服务用户和端点。参阅第 1 章 1.7 节和 1.8 节。

执行以下命令，登录到使用 Vagrant 创建的 OpenStack 控制节点。

```
vagrant ssh controller
```

操作步骤

执行如下步骤，使用 OpenStack 身份认证服务配置 OpenStack 镜像服务。

1. 首先编辑/etc/glance/glance-api.conf 文件，增加[keystone_authtoken]字段，告知 OpenStack 镜像服务使用 OpenStack 身份认证服务。注意，我们在配置里使用 insecure = True，因为我们在使用自签名的证书。在生产环境中，我们会使用发行的证书，所以不需要这个参数，代码如下。

```
[keystone_authtoken]
auth_uri = https://192.168.100.200:35357/v2.0/
identity_uri = https://192.168.100.200:5000
admin_tenant_name = service
admin_user = glance
admin_password = glance
insecure = True
```

2. 重复这个过程，编辑/etc/glance/glance-registry.conf 文件，在[keystone_authtoken]字段配置 glance 服务的用户。我们使用 insecure = True，因为我们的示例使用自签名的证书，代码如下。

```
[keystone_authtoken]
auth_uri = https://192.168.100.200:35357/v2.0/
identity_uri = https://192.168.100.200:5000
admin_tenant_name = service
```

```
admin_user = glance
admin_password = glance
insecure = True
```

3. 最后，重启这两个服务来使更改生效。

```
sudo restart glance-api
sudo restart glance-registry
```

工作原理

OpenStack 镜像服务运行两个服务。其中，glance-api 是客户端及其他服务与 glance 通信的接口，而 glance-registry 用于管理存储在硬盘和 registry 数据库中的对象。这两个服务都需要在它们的配置文件中设置好验证凭证，以方便 OpenStack 身份认证服务对其用户进行鉴权。

2.4 用 OpenStack 对象存储配置 OpenStack 镜像服务

镜像默认作为文件存储在 /var/lib/glance/images/ 这个目录。但是，OpenStack 镜像服务能配置为使用 OpenStack 对象存储来存储镜像，也支持使用其他后端镜像，诸如 Ceph 和 GlusterFS。在本节里，我们将会浏览配置 OpenStack 镜像服务 (Glance) 来使用对象存储服务 (Swift) 所需的步骤。

准备工作

首先，确保已经登录到 OpenStack 控制主机，或者登录到运行 OpenStack 镜像服务的主机上。

要登录通过 Vagrant 创建的 OpenStack 控制主机，执行以下命令。

```
vagrant ssh controller
```

操作步骤

执行以下步骤，配置 OpenStack 镜像服务来使用 OpenStack 对象存储。

1. 编辑 /etc/glance/glance-api.conf，告知 Glance 我们将使用 Swift 而不是默认的文件系统，在 [DEFAULT] 字段下编辑下面这一行。

```
[DEFAULT]
default_store = swift
```

2. 接下来在相同文件中编辑 [glance_store] 字段，配置 Swift。

```
[glance_store]
stores = glance.store.filesystem.Store,
```

```

glance.store.http.Store,
glance.store.swift.Store
swift_store_auth_version = 2
swift_store_auth_address = https://192.168.100.200:5000/v2.0/
swift_store_user = service:glance
swift_store_key = glance
swift_store_container = glance
swift_store_create_container_on_put = True
swift_store_large_object_size = 5120
swift_store_large_object_chunk_size = 200
swift_enable_snet = False
swift_store_auth_insecure = True

```



我们使用 `swift_store_auth_insecure = True`，因为我们为 SSL Keystone 的实现使用自签名的证书。根据你的环境调整这个设置。

3. 最后，重启两个 OpenStack 镜像服务进程，使改动生效。

```

sudo restart glance-api
sudo restart glance-registry

```

工作原理

OpenStack 镜像服务可以配置为使用诸多不同的后端来存储镜像，如 Ceph、OpenStack 对象存储和磁盘。一旦对象存储被配置为后端，镜像就会被上传到 Swift，而不是存储在本地。参阅第 5 章中关于配置和使用对象存储部分。

2.5 用 OpenStack 镜像服务管理镜像

通过 `glance` 命令行工具，可以在 OpenStack 存储中上传和管理镜像。它可以让我们上传、删除、修改 OpenStack 环境中使用的存储镜像的相关信息。

准备工作

首先，请确认登录到了可以运行 `glance` 工具的 Ubuntu 客户机或者直接运行 OpenStack 镜像服务的 OpenStack 控制节点上。如果没有安装 Glance 客户端，可以通过下面的方法安装。

```

sudo apt-get update
sudo apt-get install python-glanceclient

```

为了保证环境变量设置正确，`admin` 用户和密码应和之前创建的保持一致，执行以下操作。

```

export OS_TENANT_NAME=cookbook

```



```
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

可以有多种方式上传和查看 OpenStack 镜像服务中的镜像文件。按照下面的步骤上传镜像文件和查看上传镜像的详细信息。

上传 Ubuntu 镜像文件

Ubuntu 提供的镜像可以方便地添加到 OpenStack 环境之中。

1. 首先, 从 <http://uec-images.ubuntu.com> 上下载 Ubuntu 云系统镜像。

```
wget https://cloud-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-disk1.img
```

2. 然后上传这个文件。

```
glance image-create \
  --name='Ubuntu 14.04 x86_64 Server' \
  --disk-format=qcow2 \
  --container-format=bare \
  --is-public True < \
  trusty-server-cloudimg-amd64-disk1.img
```

输出如图 2-2 所示。

Property	Value
checksum	d03071f4d387dfb976e29f00ff397496
container_format	bare
created_at	2015-01-09T09:31:35
deleted	False
deleted_at	None
disk_format	qcow2
id	18584bff-2c12-4c2d-85f6-59771073c936
is_public	False
min_disk	0
min_ram	0
name	Ubuntu 14.04 x86_64 Server
owner	45c787efeaec42aa9cab522711bf5f4d
protected	False
size	256180736
status	active
updated_at	2015-01-09T09:31:37
virtual_size	None

图 2-2

列出镜像文件

要列出 OpenStack 镜像服务资源库中的镜像文件, 可以直接使用 Glance 客户端来询问

镜像服务，或使用管理 OpenStack 环境的 Nova 客户端，这将在第 4 章中详细介绍。

要列出用户可用的镜像，可使用下面的命令。

```
glance image-list
```

输出结果如图 2-3 所示。

ID	Name	Disk Format	Container Format	Size	Status
fc1ec7e2-f9ef-4afa-9634-993e370a26c0	cirros-image	qcow2	bare	9761280	active
db02ab51-f9a1-4e38-8c3d-22b367962154	trusty-image	qcow2	bare	256115200	active
18584bff-2c12-4c2d-85f6-59771073c936	Ubuntu 14.04 x86_64 Server	qcow2	bare	256180736	active

图 2-3

查看镜像文件详细信息

可以查看资源库中有关镜像的更详细信息。该信息可以通过下面的命令获得。

```
glance image-show IMAGE_ID
```

例如：

```
glance image-show 18584bff-2c12-4c2d-85f6-59771073c936
```

它返回的详细信息，与上传镜像时输出的内容相同。

删除镜像文件

在一个 OpenStack 云计算环境中，将会有很多情况下需要删除已有的镜像文件。只要具备相应权限，即可以通过下面的方式删除私有或公共的镜像文件。

1. 使用如下命令删除镜像文件。

```
glance image-delete IMAGE_ID
```

例如：

```
glance image-delete 794dca52-5fcd-4216-ac8e-7655cdc88852
```

2. 当成功执行删除镜像后，OpenStack Image 不会产生输出。可以通过执行 `glance image-list` 验证结果。

将私有镜像文件设为公开镜像文件

当上传镜像文件时，这些镜像文件将默认只有上传者才拥有权限，即私有镜像文件。如果使用上述方式上传了镜像文件，但又希望它可以给其他用户使用，在 OpenStack 镜像服务下，使用下面的方法将其设为公开。

1. 首先, 列出并查看镜像文件确认哪一个需要公开。在本书的例子中, 选择了最初上传的镜像。

```
glance image-show IMAGE_ID
```

例如:

```
glance image-show 18584bff-2c12-4c2d-85f6-59771073c936
```

输出结果如图 2-4 所示。

Property	Value
checksum	d03071f4d387dfb976e29f00ff397496
container_format	bare
created_at	2015-01-09T09:31:35
deleted	False
deleted_at	None
disk_format	qcow2
id	18584bff-2c12-4c2d-85f6-59771073c936
is_public	True
min_disk	0
min_ram	0
name	Ubuntu 14.04 x86_64 Server
owner	45c787efeac42aa9cab522711bf5f4d
protected	False
size	256180736
status	active
updated_at	2015-01-09T09:41:21
virtual_size	None

图 2-4

2. 这时, 可以将其设为公开镜像, 使云环境内所有用户均可以使用这个镜像文件。

```
glance image-update 18584bff-2c12-4c2d-85f6-59771073c936 \
--is-public True
```

3. 列出可用的公开镜像。

```
glance image-show 18584bff-2c12-4c2d-85f6-59771073c936
```

输出如图 2-5 所示。

Property	Value
checksum	d03071f4d387dfb976e29f00ff397496
container_format	bare
created_at	2015-01-09T09:31:35
deleted	False
deleted_at	None
disk_format	qcow2
id	18584bff-2c12-4c2d-85f6-59771073c936
is_public	True
min_disk	0
min_ram	0
name	Ubuntu 14.04 x86_64 Server
owner	45c787efeac42aa9cab522711bf5f4d
protected	False
size	256180736
status	active
updated_at	2015-01-09T09:41:21
virtual_size	None

图 2-5

工作原理

在我们的私有云环境中，OpenStack 镜像服务是一个非常灵活的镜像管理系统，它允许用户使用多种镜像管理方式，从添加新镜像、删除镜像到更新信息，比如文件的命名方式，它让用户能很方便地识别这些镜像文件，还能将私有镜像转换为共有镜像。当然，还可以将共有镜像转换为私有镜像。

要做到这一切，只需要在任何已连接的客户端上使用 glance 工具就可以了。

2.6 注册远程存储的镜像

OpenStack 镜像服务提供了一种机制，它可以远程添加一个存储在外部位置的镜像文件。利用这种机制，可以很方便地实现在私有云中使用第三方服务器中上传的镜像文件。

准备工作

首先，确认已经登录到一台 Ubuntu 客户机，且可以运行 glance 工具。如果没有该工具，可通过以下方法安装。

```
sudo apt-get update
sudo apt-get install python-glanceclient
```

为了保证环境变量设置正确，admin 用户和密码应和之前创建的保持一致。执行以下操作。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

参照下列步骤，将远程存储镜像文件添加到 OpenStack 镜像服务中。

1. 为了注册一个远程虚拟镜像，这里使用了 location 参数指定镜像文件的位置，而不是之前通过管道的方式指定镜像文件。

```
glance image-create \
  --name='Ubuntu 12.04 x86_64 Server' \
  --disk-format=qcow2 \
  --container-format=bare \
  --public \
```

```
--location http://webserver/precise-server-cloudimg-amd64-disk1.img
```

2. 该命令返回类似于图 2-6 所示的信息，它紧接着被存储到 OpenStack 镜像服务里。

Property	Value
container_format	bare
created_at	2015-01-09T09:47:45
deleted	False
disk_format	qcow2
id	b03eb63b-e58c-4743-8dee-320b3b30fa3e
is_public	False
min_disk	0
min_ram	0
name	Ubuntu 12.04 x86_64 Server
owner	45c787efeaec42aa9cab522711bf5f4d
protected	False
size	262865408
status	active
updated_at	2015-01-09T09:47:45

图 2-6

工作原理

使用 glance 工具，可以方便快捷地将远程镜像文件添加到这个 OpenStack 镜像服务资源库中。这个方法得以实现，主要在于 location 参数的灵活性。同时还可以像指定本地镜像文件那样，设定其他元数据。

2.7 租户间共享镜像

当一个镜像是私有的时候，租户中只有上传者才能访问该镜像。OpenStack 镜像服务提供了一种机制，可以让私有镜像在不同的租户之间共享。这实现了对镜像更好的控制，不必向所有租户公开即可在多个租户之间共享镜像。

准备工作

首先，确认已经登录到一台 Ubuntu 客户机，且可以运行 glance 工具。如果没有该工具，可通过以下方法安装。

```
sudo apt-get update
sudo apt-get install glance-client
```

为了保证环境变量设置正确，admin 用户和密码应和之前创建的保持一致。执行以下操作。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
```

```
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

执行下列步骤，将 cookbook 租户中的私有镜像文件共享给其他租户。

1. 首先需要获取能够使用镜像租户的 ID，命令如下。

```
keystone tenant-list
```

2. 然后列出镜像。

```
glance image-list
```

3. 对于我们的 cookbook 租户来说，其 ID 为 45c787efeaec42aa9cab522711bf5f4d，有一个镜像，其 ID 为 18584bff-2c12-4c2d-85f6-59771073c936，可以通过以下命令共享该镜像。

```
glance member-create \
    18584bff-2c12-4c2d-85f6-59771073c936 \
    45c787efeaec42aa9cab522711bf5f4d
```

工作原理

glance 命令中的 member-create 选项可以用来与其他租户共享镜像，语法如下。

```
glance [--can-share] member-create image-id tenant-id
```

该命令有一个可选参数 --can-share，可以指定哪个租户可以共享该镜像。

2.8 查看共享镜像

当使用 member-create 选项时，可以查看为某个租户共享了哪些镜像。这可以让我们在 OpenStack 环境中管理和控制哪些用户可以访问什么类型的镜像。

准备工作

首先，确认已经登录到一台 Ubuntu 客户机，且可以运行 glance 工具。如果没有该工具，可通过以下方法安装。

```
sudo apt-get update
sudo apt-get install python-glanceclient
```

为了保证环境变量设置正确，admin 用户和密码应和之前创建的保持一致。执行以下操作。

```
export OS_TENANT_NAME=cookbook
```



```
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

执行以下步骤，查看共享给某个租户的镜像。

1. 首先需要获取租户 ID，命令如下。

```
keystone tenant-list
```

2. 然后使用租户 ID，列出已经共享给租户的镜像。

```
glance member-list --tenant-id \
45c787efeaec42aa9cab522711bf5f4d
```

3. 输出如图 2-7 所示。

Image ID	Member ID	Can Share
18584bff-2c12-4c2d-85f6-59771073c936	45c787efeaec42aa9cab522711bf5f4d	

图 2-7

工作原理

glance 命令中的 member-list 选项可以用来查看哪些镜像被共享给其他租户，语法如下。

```
glance member-list --image-id IMAGE_ID
glance member-list --tenant-id TENANT_ID
```

2.9 使用镜像元数据

我们可以设置任意的元数据，用来描述镜像和它们如何与其他 OpenStack 组件关联。这个特殊的数据，可以在镜像创建或更新时设置，用来启用其他 OpenStack 服务中的特定功能，或者仅仅只是对镜像进行自定义描述。

准备工作

首先，确认已经登录到一台 Ubuntu 客户机，且可以运行 glance 工具。如果没有该工具，可通过以下方法安装。

```
sudo apt-get update
```

```
sudo apt-get install python-glanceclient
```

为了保证环境变量设置正确，admin 用户和密码应和之前创建的保持一致，执行以下操作。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

镜像元数据可以增加、更新和删除，也可以用作主机的调度。

更新镜像的属性

执行以下步骤，更新镜像的元数据。

1. 首先获取我们想要更新元数据的镜像 ID，命令如下。

```
glance image-list
```

2. 给镜像增加元数据，设置 image_state 和 os_distro 属性。

```
glance image-update db02ab51-f9a1-4e38-8c3d-22b367962154
--property image_state=available \
--property os_distro=ubuntu
```

3. 得到图 2-8 所示的输出。

Property	Value
Property 'image_state'	available
Property 'os_distro'	ubuntu
checksum	4a992ed9b91ddea133201cd45f127156
container_format	bare
created_at	2015-01-09T05:34:09
deleted	False
deleted_at	None
disk_format	qcow2
id	db02ab51-f9a1-4e38-8c3d-22b367962154
is_public	False
min_disk	0
min_ram	0
name	trusty-image
owner	45c787efeaeac42aa9cab522711bf5f4d
protected	False
size	256115200
status	active
updated_at	2015-01-09T10:06:11
virtual_size	None

图 2-8

删除镜像的所有属性

执行以下步骤，删除镜像属性。

1. 首先获取我们想要删除元数据的镜像 ID，命令如下。

```
glance image-list
```

2. 删除所有的元数据信息，命令如下。

```
glance image-update db02ab51-f9a1-4e38-8c3d-22b367962154 \
--purge-props
```

删除镜像的特定属性

执行以下步骤，删除镜像属性。

1. 首先获取我们想要删除元数据的镜像 ID，命令如下。

```
glance image-list
```

2. 删除特定元数据时，我们需要指定哪些属性需要保留。在这个例子里，我们将会删除 `image_state` 和其他设置的属性，但是保留 `os_distro` 这个属性。请注意，用户增加属性如果没有被指定的话将会被删除，代码如下。


```
glance image-update db02ab51-f9a1-4e38-8c3d-22b367962154 \
--purge-props --property os_distro=Ubuntu
```

使用元数据做主机调度

元数据可用于决定主机的调度。举个例子，针对不同管理程序类别的主机，可以指定属性，确定镜像部署在哪一个管理程序上。执行以下步骤，使我们能基于镜像元数据调度主机。

1. 编辑 `/etc/nova/nova.conf` 文件，更新调度器属性。

```
# Scheduler
scheduler_default_filters=ImagePropertiesFilter
```

 虽然调度是一个固有的 OpenStack 计算功能，但是为了完备性，我们对 Glance 使用这个字段的配置。

2. 重启 nova 调度器。

```
sudo stop nova-scheduler
sudo start nova-scheduler
```

3. 获取我们想要更新元数据的镜像 ID。

```
glance image-list
```

4. 设置 `architecture` 和 `hypervisor` 类型属性，`kvm` 和 `qemu` 的管理程序类型都是 `qemu`。

```
glance image-update db02ab51-f9a1-4e38-8c3d-22b367962154 \
--property architecture=arm \
--property hypervisor_type=qemu
```


工作原理

glance 命令里的 glance image-update 选项允许我们增加、修改和删除自定义镜像属性。语法如下。

```
glance image-create [other options] --property <key=value>
glance image-update IMAGE_ID --property <key=value>
```

延伸阅读

□ 参见第 4 章。

2.10 迁移 VMware 镜像

我们可以把基于 VMware 的镜像 vmdk，迁移成其他磁盘镜像格式。通过一个镜像转换工具即可完成。这个工具也可以用于验证转换工作是否正常。一旦镜像转换完成，就可以把它上传到 OpenStack 镜像服务里了。

准备工作

首先，请确认已经登录到 Ubuntu 客户机，在那里我们能完成镜像的转换。确保你已安装 qemu-util，如果尚未安装，请执行如下命令。

```
sudo apt-get install qemu-utils
```

操作步骤

执行以下步骤，把 VMDK 镜像转换为 QCOW2 格式。

1. 通过如下命令确认镜像。

```
qemu-img info custom-iso-1415990568-disk1.vmdk
```

2. 输出如下。

```
image: custom-iso-1415990568-disk1.vmdk
file format: vmdk
virtual size: 39G (41943040000 bytes)
disk size: 2.7G
cluster_size: 65536
```

Format specific information:

```
cid: 2481477841
parent cid: 4294967295
create type: monolithicSparse
extents:
[0]:
```

```

virtual size: 41943040000
filename: custom-iso-1415990568-disk1.vmdk
cluster size: 65536
format:

```

3. 使用如下命令转换镜像。

```

qemu-img convert -f vmdk -O qcow2 -c \
-p custom-iso-1415990568-disk1.vmdk \
custom-iso-1415990568-disk1.qcow2

```

在这里，`-f` 指定输入磁盘镜像格式，`-o` 指定输出格式，`-c` 目标应该只被压缩成 QCOW 格式，`-p` 显示进度。

4. 通过如下命令验证转换的镜像，如果所有都跟预料一样的话，应该显示镜像是完全相同的。

```

qemu-img compare -s -f vmdk \
-F qcow2 custom-iso-1415990568-disk1.vmdk \
custom-iso-1415990568-disk1.qcow2
Images are identical.

```

工作原理

`qemu-img convert` 命令行工具能转换多种格式，包括 VMDK。应该也支持转换成 VMDK 或其他格式。QCOW 格式支持镜像的压缩，所以能得到更小的镜像，后续才有增长空间。

2.11 创建 OpenStack 镜像

现在我们可以创建自定义的 OpenStack 镜像，但是明智的做法是在 OpenStack 安装之外进行。另外，需要确保在创建镜像的系统上，没有运行 VirtualBox、Fusion 或其他类似虚拟化技术。我们将会创建一个基于 KVM 的 CentOS 镜像。

准备工作

开始之前，确保已登录一个 Linux 系统，它不是你的 OpenStack 环境。

在 Ubuntu 上，安装 `kvm/qemu` 和 `libvirt` 库。

```
sudo apt-get install qemu-kvm libvirt-bin virt-manager
```

通过如下命令启动 `libvirt-bin` 服务。

```
sudo start libvirt-bin
```

在 CentOS 或 RHEL 上命令如下。

```

sudo yum groupinstall "Virtualization" "Virtualization Platform"
sudo chkconfig libvirtd on

```

```
sudo service libvirtd start
```

在 Fedora 上命令如下。

```
sudo yum groupinstall "Virtualization" "Virtualization Platform"
sudo systemctl enable libvirtd
sudo systemctl start libvirtd
```

理想状况下，你还需要一个 VNC 客户端，不过我们的例子不需要也可以完成。

操作步骤

通过如下步骤创建一个自定义镜像。

1. 创建一个名为 openstack.txt 的 kickstart 文件。

```
install
text
url --url http://mirror.rackspace.com/CentOS/6.6/os/x86_64/
lang en_US.UTF-8
keyboard us
network --onboot yes --bootproto dhcp --noipv6
timezone --utc America/Chicago
zerombr
clearpart --all --initlabel
bootloader --location=mbr --append="crashkernel=auto rhgb quiet"
part / --fstype=ext4 --size=1024 --grow
authconfig --enableshadow --passalgo=sha512
rootpw openstack
firewall --disable
selinux --disabled
skipx
shutdown
%packages
@core
openssh-server
openssh-clients
wget
curl
git
man
vim
ntp
%end
%post
%end
```

2. 执行以下命令。

```
sudo virt-install --virt-type kvm --name centos-6.6 --ram 1024 \
--location=http://mirror.rackspace.com/CentOS/6.6/os/x86_64/ \
--disk path=/tmp/centos-6.6-vm.img,size=5 \
--network network=default --graphics vnc,listen=0.0.0.0 \
--noautoconsole --os-type=linux --os-variant=rhel6 \
--initrd-inject=centos-6.6-x86_64-openstack.txt \
```



```
--extra-args="noverifyssl console=tty0 console=ttyS0,115200 \
ks=file:/centos-6.6-x86_64-openstack.txt "
```

3. 你将会得到类似于如下的输出。

```
Starting install...
```

```
Retrieving file .treeinfo...
```

```
| 728 B 00:00 ...
```

```
Retrieving file vmlinuz...
```

```
| 7.9 MB 00:00 ...
```

```
Retrieving file initrd.img...
```

```
| 66 MB 00:00 ...
```

```
Creating domain...
```

```
| 0 B 00:01
```

```
Domain installation still in progress. You can reconnect to the
console to complete the installation process.
```

4. 通过 VNC 验证镜像的创建是否完成。VNC server 将会默认运行在 host:5900，如图 2-9 所示。

```
QEMU (centos-6.6) - VNC Viewer
mice: PS/2 mouse device common for all mice
Input: AT Translated Set 2 keyboard as /devices/platform/i8042/serio0/input/input2
rtc_cmos 00:01: RTC can wake from S4
rtc_cmos 00:01: rtc core: registered rtc_cmos as rtc0
rtc0: alarms up to one day, 114 bytes nvram
cpuidle: using governor ladder
cpuidle: using governor menu
EFI Variables Facility v0.08 2004-May-17
usbcore: registered new interface driver hiddev
usbcore: registered new interface driver usbhid
usbhid: v2.6:USB HID core driver
GRE over IPo4 demultiplexor driver
TCP cubic registered
Initializing XFRM netlink socket
NET: Registered protocol family 17
registered taskstats version 1
rtc_cmos 00:01: setting system clock to 2014-12-08 04:56:29 UTC (1418014589)
Initializing network drop monitor service
Freeing unused kernel memory: 1292k freed
Write protecting the kernel read-only data: 10240k
Freeing unused kernel memory: 788k freed
Freeing unused kernel memory: 1568k freed
System halted.
```

图 2-9

如果你无法访问 VNC 客户端，或者因为某些原因无法使用它，等待 10 分钟（估计）并且跳到下一步骤。我们将会假设它能工作。

5. 通过以下命令列出 virsh 里正在运行的虚拟机。

```
sudo virsh list --all
```

Id	Name	State
62	centos-6.6	running

6. 通过以下命令停止（销毁）虚拟机。

```
sudo virsh destroy centos-6.6
```

```
Domain centos-6.6 destroyed
```

7. 通过以下命令启动虚拟机。

```
sudo virsh start centos-6.6
Domain centos-6.6 started
```

8. 通过用户名 root 密码 openstack 登录 VM 控制台。要退出控制台会话，按 Ctrl+] 组合键。

```
sudo virsh console centos-6.6
Connected to domain centos-6.6
Escape character is ^]
```

按 Enter 键退出。

```
CentOS release 6.6 (Final)
Kernel 2.6.32-504.el6.x86_64 on an x86_64
```

```
localhost.localdomain login: root
Password:
```

9. 在虚拟机 guest 上，通过以下命令安装 cloud-init 包。

```
sudo yum install http://dl.fedoraproject.org/pub/epel/6Server/
x86_64/epel-release-6-8.noarch.rpm
```

```
sudo yum install cloud-init cloud-utils cloud-utils-growpart
```

10. 通过如下命令修改 guest 的云配置文件/etc/cloud/cloud.cfg。

```
rm /etc/cloud/cloud.cfg
vi /etc/cloud/cloud.cfg
```

11. 粘贴如下内容。

```
users:
- default

disable_root: 1
ssh_pwauth: 0

locale_configfile: /etc/sysconfig/i18n
mount_default_fields: [~, ~, 'auto', 'defaults,nofail', '0', '2']
resize_rootfs_tmp: /dev
ssh_deletekeys: 0
ssh_genkeytypes: ~
syslog_fix_perms: ~

cloud_init_modules:
- bootcmd
- write-files
- resizefs
- set_hostname
- update_hostname
- update_etc_hosts
- rsyslog
```

```

- users-groups
- ssh

cloud_config_modules:
- mounts
- locale
- set-passwords
- timezone
- puppet
- chef
- salt-minion
- mcollective
- disable-ec2-metadata
- runcmd

cloud_final_modules:
- rightscale_userdata
- scripts-per-once
- scripts-per-boot
- scripts-per-instance
- scripts-user
- ssh-authkey-fingerprints
- keys-to-console
- phone-home
- final-message

system_info:
  distro: rhel
  default_user:
    name: centos
    lock_passwd: True
    shell: /bin/bash
    sudo: ["ALL=(ALL) NOPASSWD: ALL"]
  paths:
    cloud_dir: /var/lib/cloud
    templates_dir: /etc/cloud/templates
    ssh_svcname: sshd

```

12. 通过以下命令确保 guest 能与元数据服务通信。

```
sudo echo "NOZEROCONF=yes" >> /etc/sysconfig/network
```

13. 通过以下命令删除持久规则。

```
sudo rm -f /etc/udev/rules.d/70-persistent-net.rules
```

14. 删除机器特定的 MAC 地址和 UUID。编辑/etc/sysconfig/network-scripts/ifcfg-eth0 文件，删除以 HWADDR 和 UUID 开头的行。

```
sudo sed -i '/HWADDR/d' /etc/sysconfig/network-scripts/ifcfg-eth0
sudo sed -i '/UUID/d' /etc/sysconfig/network-scripts/ifcfg-eth0
```

15. 做了上述修改之后，/etc/sysconfig/network-scripts/ifcfg-eth0 文件看起来应该像下面这样子。

```
DEVICE="eth0"
```



```
BOOTPROTO="dhcp"
IPV6INIT="no"
MTU="1500"
NM_CONTROLLED="yes"
ONBOOT="yes"
TYPE="Ethernet"
```

16. 通过以下命令清理 yum、日志文件、临时文件和历史。

```
sudo yum clean all
sudo rm -rf /var/log/*
sudo rm -rf /tmp/*
sudo history -c
```

17. 通过以下命令关掉 guest。

```
sudo shutdown -h now
```

18. 通过以下命令压缩新创建的镜像。

```
sudo qemu-img convert -c /tmp/centos-6.6-vm.img \
-O qcow2 /tmp/centos-6.6.img
```

19. 通过以下命令把镜像上传给 Glance。

```
glance image-create --name centos-6.6 \
--disk-format=qcow2 --container-format=bare --file /tmp/centos-6.6.img
```

工作原理

`qemu-img convert` 命令行工具能转换多种格式，包括 VMDK。应该也支持转换成 VMDK 或其他格式。QCOW 格式支持镜像的压缩，所以能得到更小的镜像，后续才有增长空间。

第 3 章

Neutron——OpenStack 网络服务

本章将讲述以下内容：

- ☐ 在专属网络节点安装 Neutron 和 Open vSwitch
- ☐ 配置 Neutron 和 Open vSwitch
- ☐ 安装并配置 Neutron API 服务
- ☐ 创建租户 Neutron 网络
- ☐ 删除 Neutron 网络
- ☐ 创建外部浮动 IP Neutron 网络
- ☐ 将 Neutron 网络用作不同用途
- ☐ 配置分布式虚拟路由
- ☐ 使用分布式虚拟路由

3.1 简介

OpenStack 网络服务是 OpenStack 的软件定义网络（Software Defined Networking, SDN）组件，即 Neutron。通过 SDN，我们可以定义安全的、多租户环境下的复杂网络，这解决了 Flat 网络和 VLAN 网络经常碰到的问题。在 OpenStack 中，SDN 是一个可插拔架构，意味着能够插入并管控多个交换机、防火墙和负载均衡器，实现多种功能，如防火墙即服务（Firewalls-as-a-Service）。上述一切都定义在软件中，实现了对整个云基础设施的精细化控制。

OpenStack 网络服务可以替代 OpenStack 计算服务自带的网络组件 nova-network。尽管

有人仍认为 nova-network 使用起来更加强大、可用性更高，但是许多人已经开始在生产环境中部署 OpenStack 网络服务。在未来的 OpenStack 版本中，预计会废弃 nova-network。

图 3-1 所示为本章所描述的 OpenStack 架构。

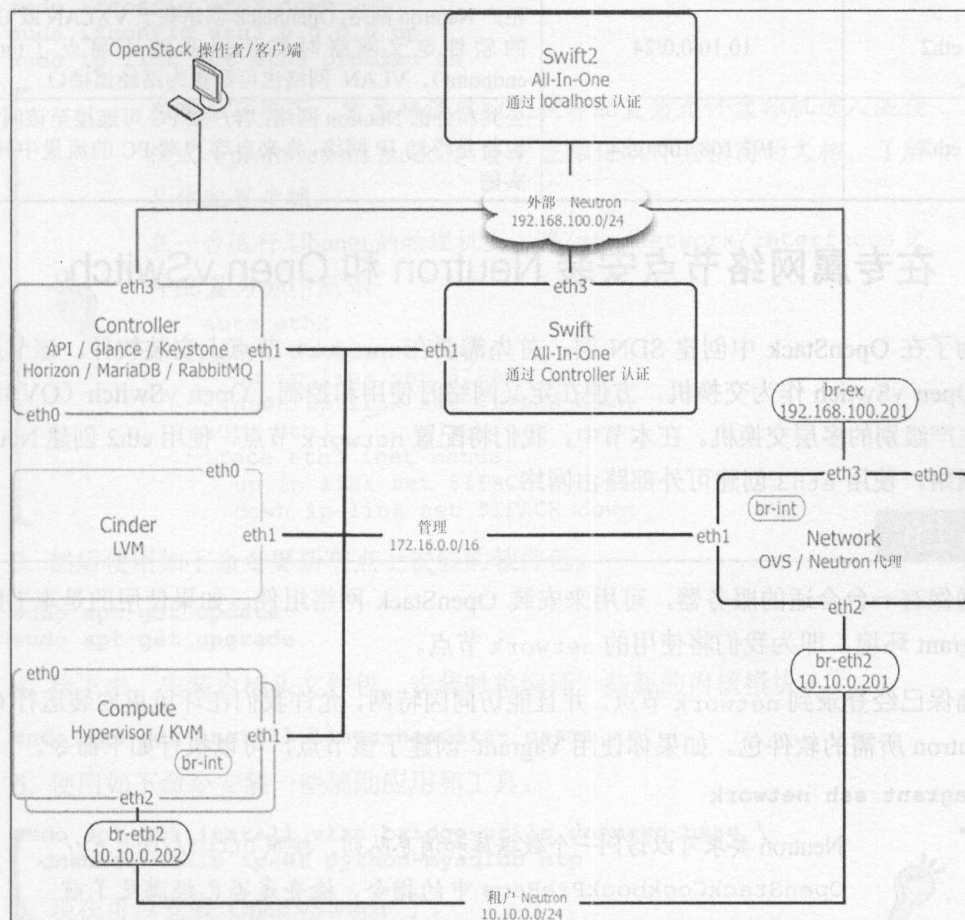


图 3-1

在这个环境中，有一个控制节点 **Controller**，一个网络主机（Network host），还有一个或多个计算主机（Compute host）。这些主机运行的都是 Ubuntu 14.04 系统，如图 3-1 所示安装有多个网卡。在本章中，将使用本书配套的虚拟环境，因此 eth0 接口将专用于虚拟环境的带外管理（out-of-band management），所以没有具体指定。在生产环境中部署 OpenStack 时，网络要求可能各不相同，应根据实际需要修改指定接口。

为了使网络配置保持一致，每个接口都关联了一个专有网络，如表 3-1 所述。

表 3-1

接口	子网络	目的
eth1	172.16.0.0/16	管理网络，用于 OpenStack 各个服务之间的内部流量
eth2	10.10.0.0/24	租户 Neutron 网络，OpenStack 创建基于 VXLAN 或 GRE 的软件定义网络时将使用其隧道端点（tunnel endpoint）。VLAN 网络也可配置为路经该接口
eth3	192.168.100.0/24	公共和外部 Neutron 网络，客户端 PC 可连接至该网络，也将是浮动 IP 网络，将来自客户端 PC 的流量中转至实例

3.2 在专属网络节点安装 Neutron 和 Open vSwitch


为了在 OpenStack 中创建 SDN 层，首先需要在 network 节点上安装软件。该节点将使用 Open vSwitch 作为交换机，方便在定义网络时使用和控制。Open vSwitch（OVS）是一个生产级别的多层交换机。在本节中，我们将配置 network 节点，使用 eth2 创建 Neutron 租户网络，使用 eth3 创建可外部路由网络。

准备工作

确保有一台合适的服务器，可用来安装 OpenStack 网络组件。如果使用的是本书配套的 Vagrant 环境，即为我们将使用的 network 节点。

确保已经登录到 network 节点，并且能访问因特网，允许我们在环境里安装运行 OVS 和 Neutron 所需的软件包。如果你使用 Vagrant 创建了该节点，可以执行如下命令。

```
vagrant ssh network
```

 Neutron 要求可以访问一个数据库和消息队列。按照 <http://bit.ly/OpenStackCookbookPreReqs> 中的指令，检查是否已经满足了前提条件。

操作步骤

执行以下步骤，配置 OpenStack 网络节点。

1. 在通过 vagrant 启动 network 节点时，必须为第三个和第四个接口（eth2 和 eth3）指定一个 IP 地址。我们不想给物理接口指定 IP 地址，但仍想让其处于 Neutron 和 OVS 的控制之下。我们随后将它们对应的地址转移到一个网桥。在图 3-1 中，这些网桥以 br-eth2 和 br-ex 表示。

2. 使用如下命令，在通过 Vagrant 创建的网络虚拟机的接口上，移除这些 IP。

```
sudo ifconfig eth2 down
sudo ifconfig eth2 0.0.0.0 up
sudo ip link set eth2 promisc on
```

```
sudo ifconfig eth3 down
sudo ifconfig eth3 0.0.0.0 up
sudo ip link set eth3 promisc on
```

在虚拟环境下，需要确保虚拟化软件配置为允许虚拟机进入混杂模式（promiscuous mode）。查看虚拟化软件提供商的文档，了解具体配置步骤。

在一台运行 Ubuntu 的物理机上，将/etc/network/interfaces 文件配置为如下所示：



```
auto eth2
iface eth2 inet manual
    up ip link set $IFACE up
    down ip link set $IFACE down
auto eth3
iface eth3 inet manual
    up ip link set $IFACE up
    down ip link set $IFACE down
```

3. 然后使用如下命令更新节点上安装的软件包。

```
sudo apt-get update
sudo apt-get upgrade
```

4. 接下来，安装内核头文件包，安装时将编译一些新的内核模块。

```
sudo apt-get install linux-headers-`uname -r`
```

5. 使用如下命令安装一些辅助应用和工具。

```
sudo apt-get install vlan bridge-utils dnsmasq-base \
    dnsmasq-utils ipset python-mysqldb ntp
```

6. 现在可以安装 Open vSwitch 了。

```
sudo apt-get install openvswitch-switch \
    openvswitch-datapath-dkms
```

7. 之后，使用如下命令启动 OVS 服务。

```
sudo service openvswitch-switch start
```

8. 现在可以开始安装运行在节点上的 Neutron 组件：Neutron DHCP 代理、Neutron L3 代理、Neutron OVS 插件以及 Neutron ML2 插件，命令如下。

```
sudo apt-get install neutron-dhcp-agent \
    neutron-l3-agent neutron-plugin-openvswitch-agent \
    neutron-plugin-ml2
```

工作原理

我们已经在 OpenStack 环境中的新节点上完成了相应包的安装, 该环境将运行 SDN 环境的软件网络组件。这其中包括通过 ML2 Neutron 插件系统实现的 OVS 服务, 以及多个与 OVS 交互的 Neutron 组件。虽然在示例中我们使用了 OVS, 但是还有许多其他第三方插件, 如 Nicira 和 Cisco UCS/Nexus。更多有关 Neutron 支持的插件的信息, 参阅 <https://wiki.openstack.org/wiki/Neutron>。

首先, 我们配置了这个交换机节点的接口, 该节点将用作租户 Neutron 和外部网络。在 OpenStack 术语中, 外部网络常被称为 Provider 网络。在数据中心的物理服务器上, 这个外部网桥接口 (br-ex) 将连接到一个可中继至其他物理服务器的网络。3.7 节将介绍如何提定该网络。Neutron 使用的两个接口在创建时都没有 IP 地址, 这样 OpenStack 环境就可以通过桥接至新的网络得到更好的控制。我们给网桥指定 IP 地址, 为这些中间存在重叠网络的 IP 端点创建隧道。在 OpenStack 中, 实例依附的是隧道内部创建的网络。

我们在 network 节点上安装了一些软件包, 列表 (除去依赖软件包) 如表 3-2 所示。

表 3-2

操作系统	linux-headers-`uname -r`
通用网络组建	vlan bridge-utils dnsmasq-base dnsmasq-utils
Open vSwitch	openvswitch-switch openvswitch-agent
Neutron	neutron-dhcp-agent neutron-l3-agent neutron-plugin-ml2 neutron-plugin-openvswitch neutron-plugin-openvswitch-agent

3.3 配置 Neutron 和 Open vSwitch

OVS 和 Neutron 的配置, 需要运行一系列 OVS 命令配置的软件交换机, 编辑多个 Neutron 配置文件。本节中, 我们将配置 network 节点。具体来说, 将使用 eth2 创建 Neutron 租户网络, 使用 eth3 创建一个可外部路由的网络。

准备工作

确保有一个适合安装 OpenStack 网络组件的服务器。如果正在使用 Vagrant 环境，我们将使用的服务器就是 network 节点。

确保登录到了 network 节点，且环境中安装了运行 OVS 和 Neutron 所需的软件包。如果是通过 Vagrant 创建的该节点，现在可以执行以下命令。

```
vagrant ssh network
```

操作步骤

执行以下步骤，在 OpenStack 的网络节点上配置 OVS 和 Neutron。

1. 安装完所需的软件包后，可以配置环境。为此，首先配置 OVS 交换机服务。需要配置一个名为 br-int 的网桥。这是一个集成网桥，用于将 SDN 环境中的所有网桥连接到一起，命令如下。

```
sudo ovs-vsctl add-br br-int
```

2. 现在配置 Neutron 租户的隧道网络网桥，可以在 OpenStack 计算主机与 network 节点之间创建 GRE 和 VXLAN 隧道，使 OpenStack 环境可使用 Neutron 的网络功能。该接口是 eth2，因此需要通过如下命令在 OVS 内配置一个叫作 br-eth2 的网桥。

```
sudo ovs-vsctl add-br br-eth2
sudo ovs-vsctl add-port br-eth2 eth2
```

3. 现在将此前分配给 eth2 接口的 IP 地址赋给该网桥。

```
sudo ifconfig br-eth2 10.10.0.201 netmask 255.255.255.0
```



这个地址位于用来创建 GRE 和 VXLAN Neutron 隧道网络的网络上。OpenStack 内的实例将把这个网络上封装的其他网络依附到 OpenStack 环境。在 vagrant 文件中指定其地址区间为 10.10.0.0/24:

```
network_config.vm.network :hostonly, "10.10.0.201",
:netmask => "255.255.255.0"
```

4. 下面添加一个在外部网络上使用的外部网桥，它将负责外部环境和 SDN 网络之间的进出流量。

```
sudo ovs-vsctl add-br br-ex
sudo ovs-vsctl add-port br-ex eth3
```

5. 现在将之前分配给 eth3 接口的 IP 地址赋给该网桥。

```
sudo ifconfig br-ex 192.168.100.201 netmask 255.255.255.0
```



这个地址所在的网络，被用于访问 OpenStack 内部的实例。在 vagrant 文件中指定其地址区间为 192.168.100.201/24:

```
network_config.vm.network :hostonly,
"192.168.100.201", :netmask => "255.255.255.0"
```

6. 确保在 /etc/sysctl.conf 文件做如下设置。

```
net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
```

7. 执行如下命令，使该文件中的系统改动生效。

```
sysctl -p
```

8. 现在需要配置后端数据库存储。先在 MariaDB 中创建名为 neutron 的数据库。执行如下命令即可（在 MariaDB 中有一个叫 root 的用户，密码为 openstack，拥有创建数据库的权限）。

```
MYSQL_ROOT_PASS=openstack
mysql -uroot -p$MYSQL_ROOT_PASS -e "CREATE DATABASE\neutron;"
```

9. 一个比较好的做法是创建一个专门用于 OpenStack 网络服务的用户，因此执行如下命令在数据库中创建一个名为 neutron 的用户。

```
MYSQL_NEUTRON_PASS=openstack
mysql -uroot -p$MYSQL_ROOT_PASS -e "GRANT ALL PRIVILEGES ON \
neutron.* TO 'neutron'@'localhost' IDENTIFIED BY \
'$MYSQL_KEYSTONE_PASS';"
mysql -uroot -p$MYSQL_ROOT_PASS -e "GRANT ALL PRIVILEGES ON \
neutron.* TO 'neutron'@'%' IDENTIFIED BY '$MYSQL_NEUTRON_PASS';"
```

10. 下一步，编辑 Neutron 的配置文件。在 network 节点上有一些配置文件要编辑。首先是 /etc/neturon/neutron.conf 文件。编辑该文件，并插入以下内容。

```
[DEFAULT]
verbose = True
debug = True
state_path = /var/lib/neutron
lock_path = $state_path/lock
log_dir = /var/log/neutron
use_syslog = True
syslog_log_facility = LOG_LOCAL0

bind_host = 0.0.0.0
bind_port = 9696

# Plugin
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

```
# auth
auth_strategy = keystone

# RPC configuration options. Defined in rpc __init__
# The messaging module to use, defaults to kombu.
rpc_backend = neutron.openstack.common.rpc.impl_kombu


rabbit_host = 172.16.0.200
rabbit_password = guest
rabbit_port = 5672
rabbit_userid = guest
rabbit_virtual_host = /
rabbit_ha_queues = false

# ===== Notification System Options =====
notification_driver = neutron.openstack.common.notifier.rpc_notifier

[agent]
root_helper = sudo

[keystone_authtoken]
auth_uri = https://192.168.100.200:35357/v2.0/
identity_uri = https://192.168.100.200:5000
admin_tenant_name = service
admin_user = neutron
admin_password = neutron
insecure = True

[database]
connection = mysql://neutron:openstack@172.16.0.200/neutron
```

 由于使用的是自签名 SSL 整数，设置 `insecure = True`。在生产环境中，需要获得有效的 SSL 整数，并设置 `insecure = False`。

11. 之后，编辑 `/etc/neutron/13_agent.ini` 文件，添加如下内容。

```
[DEFAULT]
interface_driver = neutron.agent.linux.interface.
OVSIInterfaceDriver
use_namespaces = True
```

12. 找到 `/etc/neutron/dhcp_agent.ini` 文件，插入如下内容。

```
[DEFAULT]
interface_driver = neutron.agent.linux.interface.
OVSIInterfaceDriver
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
use_namespaces = True
dnsmasq_config_file = /etc/neutron/dnsmasq-neutron.conf
```

13. 创建 `/etc/neutron/dnsmasq-neutron.conf` 文件，添加如下内容，改变 Neutron 租户接口的最大传输单位（maximum transmission unit, MTU）。


```
# To allow tunneling bytes to be appended
dhcp-option-force=26,1400
```

14. 之后, 编辑/etc/neutron/metadata_agent.ini 文件, 插入如下内容。

```
[DEFAULT]
auth_url = https://192.168.100.200:5000/v2.0
auth_region = RegionOne
admin_tenant_name = service
admin_user = neutron
admin_password = neutron
nova_metadata_ip = 172.16.0.200
metadata_proxy_shared_secret = foo
auth_insecure = True
```

15. 需要编辑的最后一个 Neutron 服务文件是/etc/neutron/plugins/ml2/ml2_conf.ini。插入如下内容。

```
[ml2]
type_drivers = gre, vxlan
tenant_network_types = vxlan
mechanism_drivers = openvswitch
```

```
[ml2_type_gre]
tunnel_id_ranges = 1:1000
```

```
[ml2_type_vxlan]
vxlan_group =
vni_ranges = 1:1000
```

```
[vxlan]
enable_vxlan = True
vxlan_group =
local_ip = 10.10.0.201
l2_population = True
```

```
[agent]
tunnel_types = vxlan
vxlan_udp_port = 4789
```

```
[ovs]
local_ip = 10.10.0.201
tunnel_type = vxlan
enable_tunneling = True
[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
enable_security_group = True
```

16. 配置好环境和交换机后, 可以重启相关服务, 使改动生效。

```
sudo service neutron-plugin-openvswitch-agent restart
sudo service neutron-dhcp-agent restart
sudo service neutron-l3-agent restart
sudo service neutron-metadata-agent restart
```

工作原理

我们完成了环境中一个新节点的配置。这个节点运行有 SDN 环境的软件网络组件。

在安装完应用和服务依赖并启动服务后，指定一个网桥作为集成网桥对环境进行配置，在内部将所有实例与其他网络连接在一起。它还将网桥连接至租户和 Provider 网络。

然后，通过编辑一些文件，在环境中启动了 Neutron 服务。第一个是 `/etc/neutron/neutron.conf` 文件。这是 Neutron 服务的主配置文件。在该文件中，定义了如何配置 Neutron，应该使用哪些组件、功能和插件。

在 `/etc/neutron/l3_agent.ini` 文件中，将指定运行租户创建重叠的 IP 区间 (`use_namespaces = True`)。这意味着，租户 A 中的用户可以创建并使用一个同时存在于租户 B 中的私有 IP CIDR。同时还指定了要使用 OVS 来提供 L3 路由功能。

在 `/etc/neutron/dhcp_agent.ini` 文件中，将指定使用 Dnsmasq 作为环境中的 DHCP 服务。同时引用了 `/etc/neutron/dnsmasq-neutron.conf` 文件，这个文件可以在 Dnsmasq 启动网络进程时传入额外的选项。这样做是为了让指定实例的网络接口中 MTU 的值为 1400。因为默认的值为 1500，与隧道给数据包添加的额外字节冲突，并且无法处理数据包分片。通过降低 MTU 的值，所有正常的 IP 信息加上额外的隧道信息就可以不需要分片，能一次性传输了。

`/etc/neutron/metadata_agent.ini` 文件告知 Neutron 和实例从何处查找元数据服务。它指向 controller 控制节点，最终也就是 nova API 服务。这里，通过 `metadata_proxy_shared_secret = foo` 设置了一个密钥，它与 controller 节点上的 `/etc/nova/nova.conf` 文件中将使用的随机关键词一致：`neutron_metadata_proxy_shared_secret=foo`。

最后一个配置文件 `/etc/neutron/plugins/ml2/ml2_conf.ini` 配置了环境中的 L2 插件，并描述了 L2 的功能。配置选项如下。

```
[ml2]
type_drivers = gre,vxlan
tenant_network_types = vxlan
mechanism_drivers = openvswitch
```

将网络类型配置为通用路由封装 (Generic Routing Encapsulation, GRE) 或 VXLAN (Virtual eXtensible LAN)。这样 SDN 环境可以通过创建的隧道封装各种协议。

指定在非管理员用户下创建私有 Neutron 网络时创建 VXLAN 隧道。管理员用户能够在命令行指定 GRE 选项。

```
[ml2_type_gre]
tunnel_id_ranges = 1:1000
```

这个选项表示当用户指定了一个私有 Neutron 租户网络，但没有指定 GRE 网络的 ID 区间

时，则从上述区间选择一个 ID。OpenStack 会确保每个创建的隧道都是唯一的，代码如下。

```
[ml2_type_vxlan]
vxlan_group =
vni_ranges = 1:1000

[vxlan]
enable_vxlan = True
vxlan_group =
local_ip = 10.10.0.201
[agent]
tunnel_types = vxlan
vxlan_udp_port = 4789
```

上述设置描述了 VXLAN 网络的选项。与 GRE 网络一样，设置一个端点 IP，即赋给隧道流量途径接口的那个 IP 地址，并指定环境中使用的有效的 vxlan ID。代码如下。

```
[ovs]
local_ip = 10.10.0.201
tunnel_type = gre
enable_tunneling = True
```

上述设置描述了要传递给 OVS 的选项，详细列举了隧道的配置。其端点地址为 10.10.0.201，并指定使用 GRE 隧道类型。代码如下。

```
[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
enable_security_group = True
```

上述代码告知 Neutron 在创建安全组时使用 IPtables 规则。



注意，在 keystone 引用行中添加了一行设置：insecure = True。
这是因为环境使用了自签名整数，这样设置之后会忽略将出现的
SSL 错误。

3.4 安装并配置 Neutron API 服务

Neutron 服务提供了一个 API，供其他服务访问和定义软件定义网络（SDN）。在本书的环境中，Neutron 服务安装在 controller 节点上，Glance 和 Keystone 等其他 API 服务也安装在该节点上。

准备工作

确保有一个适合安装 OpenStack 网络组件的服务器。如果正在使用 Vagrant 环境，将使用的服务器就是 controller 节点。

确保登录到了 controller 节点,且环境中安装了运行 OVS 和 Neutron 所需的软件包。如果是通过 Vagrant 创建的该节点,可以执行以下命令。

```
vagrant ssh controller
```



Neutron 要求可以访问一个数据库和消息队列。请检查已经满足了前提条件,具体操作步骤请参照 <http://bit.ly/OpenStackCookbook> PreReqs。

操作步骤

执行以下步骤,配置 controller 节点。

1. 首先更新节点上安装的软件包。

```
sudo apt-get update
sudo apt-get upgrade
```

2. 现在可以使用以下命令,安装 Neutron 服务和 ML2 插件。

```
sudo apt-get install neutron-server \
    neutron-plugin-ml2 ntp
```

3. 接下来,编辑 Neutron 配置文件。由于只需要提供 Neutron API 服务,首先需要在 `/etc/neutron/neutron.conf` 文件中配置服务。编辑该文件,插入下面的内容,这些内容与 network 节点上的配置相同。

```
[DEFAULT]
verbose = True
debug = True
state_path = /var/lib/neutron
lock_path = $state_path/lock
log_dir = /var/log/neutron
use_syslog = True
syslog_log_facility = LOG_LOCAL0
```

```
bind_host = 0.0.0.0
bind_port = 9696
```

```
# Plugin
```

```
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

```
# auth
```

```
auth_strategy = keystone
```

```
# RPC configuration options. Defined in rpc __init__
```

```
# The messaging module to use, defaults to kombu.
```

```
rpc_backend = neutron.openstack.common.rpc.impl_kombu
```

```
rabbit_host = 172.16.0.200
```

```
rabbit_password = guest
```

```
rabbit_port = 5672
```

```

rabbit_userid = guest
rabbit_virtual_host = /
rabbit_ha_queues = false

# ===== Notification System Options =====
notification_driver = neutron.openstack.common.notifier.rpc_notifier

# ===== neutron nova interactions =====
notify_nova_on_port_status_changes = True
notify_nova_on_port_data_changes = True
nova_url = http://172.16.0.200:8774/v2
nova_region_name = RegionOne

nova_admin_username = nova
nova_admin_tenant_name = service
nova_admin_password = nova
nova_admin_auth_url = https://192.168.100.200:35357/v2.0
nova_ca_certificates_file = /etc/ssl/certs/ca.pem
[agent]
root_helper = sudo

[keystone_authtoken]
auth_uri = https://192.168.100.200:35357/v2.0/
identity_uri = https://192.168.100.200:5000
admin_tenant_name = service
admin_user = neutron
admin_password = neutron
insecure = True

[database]
connection = mysql://neutron:openstack@172.16.0.200/neutron

```

4. 然后，需要编辑/etc/neutron/plugins/ml2/ml2_conf.ini 文件，插入如下内容，这些内容与 network 节点上的配置相同（除 local_ip 选项外）。

```

[ml2]
type_drivers = gre, vxlan
tenant_network_types = vxlan
mechanism_drivers = openvswitch

[ml2_type_gre]
tunnel_id_ranges = 1:1000

[ml2_type_vxlan]
vxlan_group =
vni_ranges = 1:1000

[vxlan]
enable_vxlan = True
vxlan_group =
local_ip =
[agent]
tunnel_types = vxlan
vxlan_udp_port = 4789
[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
enable_security_group = True

```

5. 在正确配置这些文件之后，运行如下命令，确保 Neutron 数据库的状态与当前的 OpenStack 版本相匹配。

```
sudo neutron-db-manage\
  --config-file /etc/neutron/neutron.conf\
  --config-file /etc/neutron/plugins/ml2/ml2_conf.ini\
  upgrade junio
```

6. 这一步，配置 Nova 服务以使用 Neutron。Nova 组件的安装将在下一章介绍，但这里为了方便也做说明。安装完 Nova 组件后，配置/etc/nova/nova.conf 文件，告知 OpenStack 计算组件使用 Neutron。在/etc/nova/nova.conf 文件的[Default]部分下添加如下设置。

```
# Network settings
network_api_class=nova.network.neutronv2.api.API
neutron_url=http://172.16.0.200:9696/
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_admin_username=neutron
neutron_admin_password=neutron
neutron_admin_auth_url=https://192.168.100.200:35357/v2.0
neutron_ca_certificates_file=/etc/ssl/certs/ca.pem
libvirt_vif_driver=nova.virt.libvirt.vif.
LibvirtHybridOVSBridgeDriver
linuxnet_interface_driver=nova.network.linux_net.
LinuxOVSIfaceDriver
firewall_driver=nova.virt.libvirt.firewall.IptablesFirewallDriver
service_neutron_metadata_proxy=true
neutron_metadata_proxy_shared_secret=foo
```

7. 使用如下命令，重启该节点上运行的 Neutron 服务，使得改动生效。

```
sudo service neutron-server restart
```

8. 安装 Nova 之后，重启该节点上运行的 Nova 服务，使得/etc/nova/nova.conf 文件中的改动生效。

```
ls /etc/init/nova-* | cut -d '/' -f4 | cut -d '.' -f1 | while read
S; do sudo stop $S; sudo start $S; done
```

工作原理

掌握正确的信息之后，在 controller 节点上配置 Neutron API 服务非常便捷。

只需要安装一些必须的软件包。

使用如下命令安装 Neutron 软件包。

```
neutron-server
neutron-plugin-ml2
```

安装 Neutron 软件包后，配置/etc/neutron/neutron.conf 文件，其内容与网络节点上的文件配置相同，只是新增了一个部分：Neutron 与 Nova 交互。这里，确保正确设

置，允许 Nova 与 Neutron 相互操作。还配置了 ML2 插件文件，其内容与 network 节点上的相同，但是这里省略了 OVS 部分，因为在 controller 节点上这个设置是多余的。

然后运行一个命令，确保 Neutron 数据库的行、列数正确，与 OpenStack Juno 发布版匹配。

最后，配置/etc/nova/nova.conf 文件。对于 OpenStack 计算服务来说，这是最重要的配置文件。

- ❑ `network_api_class=nova.network.neutronv2.api.API`: 告知 OpenStack 计算服务使用 Neutron 网络服务。
 - ❑ `neutron_url=http://172.16.0.200:9696/`: Neutron API 服务器的地址（运行在 controller 节点上）。
 - ❑ `neutron_auth_strategy=keystone`: 告知 Neutron 使用 OpenStack 身份认证服务 Keystone。
 - ❑ `neutron_admin_tenant_name=service`: Keystone 中服务租户的名字。
 - ❑ `neutron_admin_username=neutron`: Neutron 中用于 Keystone 验证的用户名。
 - ❑ `neutron_admin_password=neutron`: Neutron 中用于 Keystone 验证的密码。
 - ❑ `neutron_admin_auth_url=https://172.16.0.200:35357/v2.0`: Keystone 服务的地址。
 - ❑ `neutron_ca_certificate_file=/etc/ssl/certs/ca.pem`: 引用了在第 1 章中生成的证书文件，不用设置 `insecure` 标志即可正常调用 Keystone 的 SSL 证书。
 - ❑ `libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtHybridOVSBridgeDriver`: 告知 Libvirt 使用 OVS 网桥驱动。
 - ❑ `linuxnet_interface_driver=nova.network.linux_net.LinuxOVSIfaceDriver`: 用来在 Linux 主机上创建 Ethernet 设备的驱动。
 - ❑ `firewall_driver=nova.virt.libvirt.firewall.IptablesFirewallDriver`: 用来管理防火墙的驱动。
 - ❑ `service_neutron_metadata_proxy=true`: 可以利用元数据代理服务，将 Neutron 的请求转发给 Nova API 服务。
 - ❑ `foo`: 使用代理服务前需设置的随机密钥。该值在所有运行该服务的节点上应保持一致，已确保转发代理请求时的安全性。
- `neutron_metadata_proxy_shared_secret=foo`

延伸阅读

□ 参见第 4 章。

3.5 创建租户 Neutron 网络

现在已经成功运行了 OpenStack 网络服务，接下来可以使用这些服务在 OpenStack 环境中创建网络。每个租户创建对应的网络，并使用这些网络连接至虚拟机。Neutron 网络可以是私有的，也可以是共享的。如果是私有的 Neutron 网络，只有该租户的实例和操作者才能利用这些网络。如果是共享的 Neutron 网络，所有实例均可使用该共享网络，因此要慎重使用该网络功能，确保租户之间的安全。在使用共享网络时，应实现安全组规则，确保数据流量满足安全要求。

准备工作

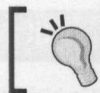
确保有一台适合使用 Neutron 的服务器。如果正在使用配套的 Vagrant 环境，则可以使用 controller 节点。该节点上安装了 python-neutronclient 软件包，提供了 neutron 命令行客户端。

如果是通过 Vagrant 创建的该节点，可以执行以下命令。

```
vagrant ssh controller
```

确保满足以下设置（如果没有使用 Vagrant 环境，调整证书和密钥文件的路径，与使用的环境相匹配）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/akey.pem
export OS_CACERT=/vagrant/ca.pem
```



这时，应该已经安装好并正确配置了 Keystone 服务。更多信息，参考第 1 章 1.2 节。

操作步骤

执行以下步骤，为特定租户创建一个私有的 Neutron 网络。

1. 首先需要获得租户 ID，为该租户创建网络时使用。执行如下命令。

```
TENANT_ID=$(keystone tenant-list\
| awk '/ \ cookbook\ / {print $2}')
```

2. 然后, 使用该值为该租户创建一个第2层网络。

```
neutron net-create\
--tenant-id ${TENANT_ID}\
cookbook_network_1
```

3. 创建好网络之后, 使用 CIDR 格式 (10.200.0.0/24) 为该网络分配一个子网。

```
neutron subnet-create \
--tenant-id ${TENANT_ID} \
--name cookbook_subnet_1 \
cookbook_network_1 \
10.200.0.0/24
```

4. 然后, 在该网络上创建一个路由器, 用作实例的默认网关。添加路由器是一个可选步骤——这是一个设计考量, 可以将流量从一个网络转发至另一个网络。该选项避免了为多宿主实例创建多个接口和网络。该路由器将用于从物理主机 IP 范围中选取一个 IP 地址, 为虚拟机实例提供网络访问。

```
neutron router-create \
--tenant-id ${TENANT_ID} \
cookbook_network_1
```

5. 将该路由器添加至子网。

```
neutron router-interface-add \
cookbook_router_1 \
cookbook_subnet_1
```

工作原理

创建一个带子网的网络, 在虚拟机启动时使用。使用如下命令, 创建一个网络。

```
neutron net-create \
--tenant-id TENANT_ID \
NAME_OF_NETWORK
```

使用如下语法, 创建一个子网。

```
neutron subnet-create \
--tenant-id TENANT_ID \
--name NAME_OF_SUBNET \
NAME_OF_NETWORK \
CIDR
```

网络上的路由器是可选的, 功能是将流量从一个子网转至另一个子网。在 Neutron SDN 网络下, 也是如此。第3层 (L3) 路由器可以用来配置网关, 按需将流量路由至其他网络。如果只要求虚拟机实例在相同子网下通信, 就没必要设置路由器, 因为没有需要路由的网络。创建路由器的命令如下。


```
neutron router-create \
  --tenant-id TENANT_ID \
  NAME_OF_ROUTER
```

将路由器添加至子网 [用于路由来自其他网络（物理或软件定义网络）的流量] 的语法如下。

```
neutron router-interface-add \
  ROUTER_NAME \
  SUBNET_NAME
```

之后，可以继续使用上述命令，向该路由器上添加更多子网，实现 OpenStack Neutron 不同子网间的流量流转。

3.6 删除 Neutron 网络

删除 Neutron 网络的操作步骤，与创建网络的步骤类似。

准备工作

确保有一台适合使用 Neutron 的服务器。如果正在使用配套的 Vagrant 环境，则可以使用 controller 控制节点。该节点上安装了 python-neutronclient 软件包，提供了 neutron 命令行客户端。

如果是通过 Vagrant 创建的该节点，可以执行以下命令。

```
vagrant ssh controller
```

确保设置好了如下证书（如果没有使用 Vagrant 环境，请调整证书和密钥文件的路径，与使用环境相匹配。）

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

执行如下步骤，删除特定租户的 Neutron 网络。

1. 使用如下命令，列出所有网络。

```
neutron net-list
```

输出如图 3-2 所示。

id	name	subnets
5b738491-4368-4e56-adaa-f4bdb0ef9dd9	cookbook_network_1	6436a0dc-1537-4010-8981-ccb34fa35ee 10.200.0.0/24

图 3-2

2. 执行如下命令，列出所有子网络。

```
neutron subnet-list
```

输出如图 3-3 所示。

id	name	cidr	allocation_pools
6436a0dc-1537-4010-8981-ccb34fa35ee	cookbook_subnet_1	10.200.0.0/24	{ "start": "10.200.0.2", "end": "10.200.0.254" }

图 3-3

3. 要删除网络和子网络，请确保没有实例或服务正在使用它们。使用如下命令，查询 Neutron 中的端口列表，检查哪些端口连接到了将删除的网络。

```
neutron port-list
```

输出如图 3-4 所示。

id	name	mac_address	fixed_ips
0ecef1b17-f126-4205-9c86-6708316d2346		fa:16:3e:e2:73:4c	{ "subnet_id": "6436a0dc-1537-4010-8981-ccb34fa35ee", "ip_address": "10.200.0.1" }
8d007e1e-fd9e-4eb4-8f94-144bff91ac96		fa:16:3e:91:66:fc	{ "subnet_id": "6436a0dc-1537-4010-8981-ccb34fa35ee", "ip_address": "10.200.0.3" }

图 3-4

4. 还可执行如下命令，查看正在运行的实例及其使用的网络。

```
nova list
```

输出如图 3-5 所示。

ID	Name	Status	Networks
0fa76731-8da5-4251-8308-0aa10f739e97	test1	ACTIVE	cookbook_network_1=10.200.0.4

图 3-5

可以看到，在将要删除的网络 `cookbook_network_1` 上已经有一个实例了。

5. 所以，需要删除运行在该网络的虚拟机实例，操作如下。

```
nova delete test1
```

6. 所有运行在该网络的虚拟机实例都已停止后，便可使用如下命令，删除连接到该网络的路由器接口。

```
ROUTER_ID=$(neutron router-list \
| awk '/\ cookbook_router_1\ / {print $2}')
```

```
SUBNET_ID=$(neutron subnet-list \
| awk '/\ cookbook_subnet_1\ / {print $2}')
```

```
neutron router-interface-delete \
  ${ROUTER_ID} \
  ${SUBNET_ID}
```

7. 删除路由器接口之后，可接着使用如下命令删除子网络。

```
neutron subnet-delete cookbook_subnet_1
```

8. 删除子网络后，可使用如下命令删除网络。

```
neutron net-delete cookbook_network_1
```

工作原理

通过上述一系列操作删除了网络。这包括首先移除所有连接到该网络的（虚拟）设备，如虚拟机实例和路由器；然后删除掉连接到网络的子网；最后删除网络本身。下面来看网络列表。

- ☐ 列出全部网络。

```
neutron net-list
```

- ☐ 列出全部子网。

```
neutron subnet-list
```

- ☐ 列出已经使用的 Neutron 端口。

```
neutron port-list
```

- ☐ 从子网删除一个路由器接口。

```
neutron router-interface-delete \
  ROUTER_ID \
  SUBNET_ID
```

- ☐ 删除子网。

```
neutron subnet-delete NAME_OF_SUBNET
```

- ☐ 删除网络。

```
neutron subnet-delete NAME_OF_NETWORK
```

3.7 创建外部浮动 IP Neutron 网络

使用 Neutron 可以很容易地创建多个私有网络，让虚拟机实例之间相互通信。但如果要从外部访问这些实例，则需要在提供商网络（外部网络）上创建一个路由器接入到

OpenStack 环境中。该提供商网络可以为实例分配浮动地址。

准备工作

确保有一台适合使用 Neutron 的服务器。如果正在使用配套的 Vagrant 环境，则可以使用 controller 控制节点。该节点上安装了 python-neutronclient 软件包，提供了 neutron 命令行客户端。

如果是通过 Vagrant 创建的该节点，可以执行以下命令。

```
vagrant ssh controller
```

确保设置好了如下证书（如果没有使用 Vagrant 环境，请调整证书和密钥文件的路径，与使用环境相匹配。）

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/akey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

在 Neutron 网络中，为一个租户创建外部路由器需要有租户管理员权限。首先，需要在 admin 租户里创建一个公共网络；然后，将它连接到需要从外部访问虚拟机实例的租户的路由器。为实例分配一个浮动 IP 地址后，便可以从外部访问该实例了。

正确设置好管理员权限后，执行如下步骤。

1. 执行如下命令，获取服务租户 service 的 ID。

```
ADMIN_TENANT_ID=$(keystone tenant-list \  
| awk '/ service / {print $2}')
```



这里不是必须使用 service 租户。只需要一个属于 admin 用户，且不是私有租户的租户。

2. 现在创建一个名为 floatingNet 的公有网络，用来提供外部路由功能。执行以下指令。

```
neutron net-create \  
--tenant-id ${ADMIN_TENANT_ID} \  
--router:external=True \  
floatingNet
```

3. 在该网络上创建一个外部/浮动地址区间。本例中的外部子网络地址区间为 192.168.100.0/24。使用以下指令，创建一个地址区间，区间内的地址将会手动分配给实例作为浮动地址。确保地址池（允许的 IP 地址列表）没有与物理环境中的任何 IP 冲突。

```
neutron subnet-create \
  --tenant-id ${ADMIN_TENANT_ID} \
  --name floatingSubnet \
  --allocation-pool \
    start=192.168.100.10,end=192.168.100.20 \
  --enable_dhcp=False \
  floatingNet \
  192.168.100.0/24
```

4. 使用如下命令，将 Cookbook 路由器（如 3.5 节的第 4 步所述）上的网关分配给该浮动网络。

```
neutron router-gateway-set \
  cookbook_router_1 \
  floatingNet
```

5. 设置完毕后，便可利用该浮动网络了。为此，将浮动 IP 分配给正在运行的虚拟机实例。首先需要通过 nova list 命令查看 cookbooko_network_1 网络上的虚拟机实例被分配的 IP。

```
nova list
```

输出如图 3-6 所示。

ID	Name	Status	Networks
9f8dff28-41fc-4f9f-a41f-a858abebc529	test1	ACTIVE	cookbookNet=10.200.0.2

图 3-6

6. 同时查看一下环境中使用的路由器和 Neutron 网络端口的信息。使用如下指令显示 cookbook_router_1 网络的信息。

```
neutron router-show cookbook_router_1
```

输出如图 3-7 所示，这里会用到其中的路由器 ID 和网络 ID。

Field	Value
admin_state_up	True
external_gateway_info	{ "network_id": "213fedde-ae5e-4396-9754-cb757cba25ea" }
id	f0a5c988-6eb2-4593-8b15-98896fd55a3a
name	cookbookRouter
routes	
status	ACTIVE
tenant_id	d856d921d02d4ded8f590e30a5392254

图 3-7

7. 执行如下命令，将一个浮动 IP 分配至连接到该端口的虚拟机实例。该命令首先从 floatingNet 网络中创建一个可使用的浮动 IP。

```
neutron floatingip-create \
  --tenant-id ${ADMIN_TENANT_ID} \
  floatingNet
```

输出如图 3-8 所示。

Created a new floatingip:

Field	Value
fixed_ip_address	192.168.100.11
floating_ip_address	2c7d13ff-f634-4d46-8165-a7c989d974b0
floating_network_id	48e2ca77-af4d-44b3-8c10-b6574d94d6ce
id	48e2ca77-af4d-44b3-8c10-b6574d94d6ce
port_id	
router_id	
status	DOWN
tenant_id	210ef2a4890a4064ba646be1e84ae1f

图 3-8

8. 将该浮动 IP 分配给虚拟机实例连接的端口。可通过查看路由器上使用的端口列表，找到端口信息。

```
neutron port-list \
--router_id=e63fe19d-7628-4180-994d-72035f770d77
```

输出如图 3-9 所示，所需的信息与 nova list 命令中列出的 IP 地址是一致的。本例中，需要的是匹配 IP 地址 10.200.0.2 的端口 ID，后者就是分配给实例的端口。

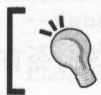
id	name	mac_address	fixed_ips
41ea7756-9521-4ba2-a885-1aca70a96ddc		fa:16:3e:b4:b4:04	{"subnet_id": "a2580694-d5f4-41b4-9ede-f5212d86deba", "ip_address": "192.168.100.10"}
5f1f68a4-2af2-4528-934d-f7f52ac5b3d3		fa:16:3e:a3:2b:6f	{"subnet_id": "e88b3347-dbd4-40c9-abf2-27762dfbb6a9", "ip_address": "10.200.0.2"}
85f1f3ad-4285-42aa-a15e-45628f865f04		fa:16:3e:33:35:16	{"subnet_id": "e88b3347-dbd4-40c9-abf2-27762dfbb6a9", "ip_address": "10.200.0.3"}
c8a2fa53-7aa8-459e-9233-2ec180049c3c		fa:16:3e:90:80:6c	{"subnet_id": "e88b3347-dbd4-40c9-abf2-27762dfbb6a9", "ip_address": "10.200.0.1"}

图 3-9

9. 在上述输出中，IP 地址为 10.200.0.2 的实例连接到了 ID 为 3e5a298b-5ca8-4484-b473-fa71410fd31c 的端口。创建浮动 IP 时，其 ID 是 48e2ca77-af4d-44b3-8c10-b6574d94d6ce。然后使用如下命令，将该浮动 IP 与实例端口 ID 关联在一起，从而将浮动 IP 分配给实例。

```
neutron floatingip-associate \
48e2ca77-af4d-44b3-8c10-b6574d94d6ce \
3e5a298b-5ca8-4484-b473-fa71410fd31c
```

命令成功后，会输出消息 Associated floating IP 48e2ca77-af4d-44b3-8c10-b6574d94d6ce。



可使用 `neutron floatingip-list` 命令查看可用浮动 IP 地址和 ID 列表。

10. 现在可以使用分配的浮动 IP 地址 192.168.100.11 访问之前只能从 network 节点访问的虚拟机实例了，如图 3-10 所示。

ID	Name	Status	Networks
9f6dff28-41fc-4f9f-a41f-a858abebc529	test1	ACTIVE	cookbookNet=10.200.0.2, 192.168.100.11

图 3-10

工作原理

本节创建了一个可以给实例分配浮动地址的网络。该子网能够从 OpenStack 之外的其他网络，或因特网上的公网地址路由进入。首先，在 admin 租户里使用 `neutron net-create` 命令和 `--router:external=True` 标志创建一个带路由器的网络。

```
neutron net-create \
  --tenant-id ADMIN_TENANT_ID \
  --router:external=True \
  NAME_OF_EXTERNAL_NETWORK
```

为了给实例手动分配浮动 IP 地址，应为子网定义一个 IP 地址区间并禁用 DHCP。

```
neutron subnet-create \
  --tenant-id ADMIN_TENANT_ID \
  --name NAME_OF_SUBNET \
  --allocation-pool start=IP_RANGE_START,end=IP_RANGE_END \
  --enable_dhcp=False \
  EXTERNAL_NETWORK_NAME \
  SUBNET_CIDR
```

接下来，使用如下指令，给该网络分配一个已有的路由器网关。该路由器会为连接到它的私有网络中的实例提供 NAT 服务。

```
neutron router-gateway-set \
  ROUTER_NAME \
  EXTERNAL_NETWORK_NAME
```

配置完成后，从刚创建的地址空间中分配一个浮动 IP 地址给运行中的实例。执行如下命令。

```
nova list
```

使用如下命令，取得实例的 IP 地址。

```
neutron router-show ROUTER_NAME
```

使用如下命令，取得路由器使用的端口。

```
neutron port-list \
  --router_id=ROUTER_ID
```

使用如下命令，从外部网络中创建一个可使用的浮动 IP（注意租户 ID 不用必须是示例中使用的 admin 租户）。

```
neutron floatingip-create \
```

```
--tenant-id=TENANT_ID \
EXTERNAL_NETWORK_NAME
```

使用实例关联的端口 ID（见路由器的端口列表），将浮动 IP 与实例关联起来。

```
neutron floatingip-associate \
    FLOATING_IP_ID \
    VM_PORT_ON_ROUTER
```

之后，即可从物理网络中使用该浮动 IP 访问该实例。

3.8 Neutron 网络的不同用途

使用 Neutron 可以轻松地创建允许实例之间进行通信的私有网络。有时用户可能要求网络具备某些特性，例如，使用独享 10G 接口时，需要为某个物理网络设备（如负载均衡或 SQL 服务器集群）创建 VLAN 网络。通过 Neutron 的 SDN（软件定义网络），管理员可以轻松创建满足这些用途的网络。

图 3-11 列出了 Neutron 中可使用的另一个接口的示例环境。环境中的 SQL 物理服务器可以在 VLAN 200 上 192.168.200.0/24 的一个子网。要在 OpenStack 内使用该网络，可以指定另一个接口 eth4，用来连接 OpenStack 虚拟实例与使用该子网和 VLAN 的物理服务器。

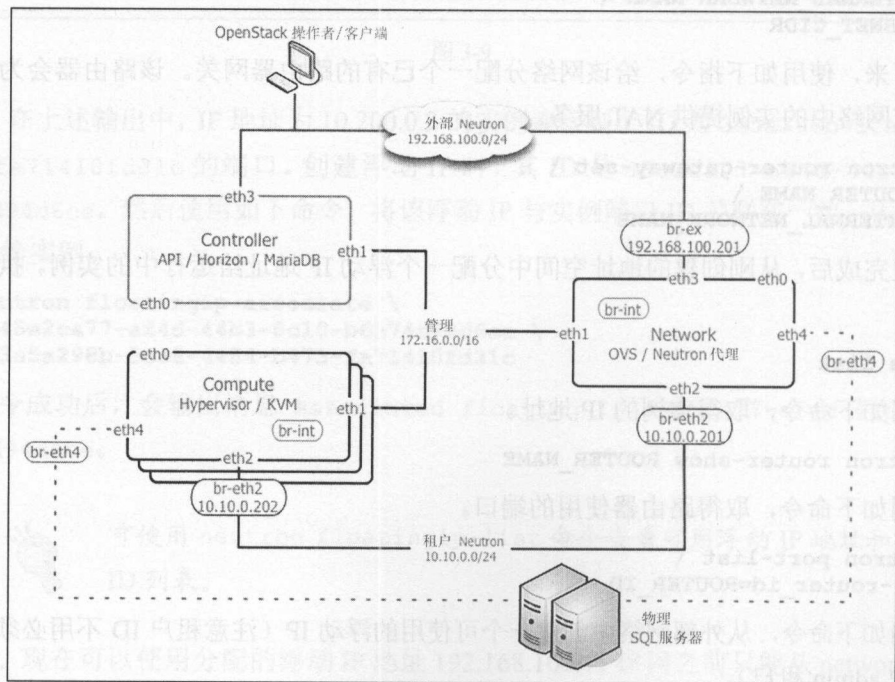


图 3-11

准备工作

确保登录到了网络和计算节点，因为我们将在两个节点上配置一个用于 OpenStack 网络的新接口。如果使用 Vagrant 创建了这些节点，可执行如下命令。

```
vagrant ssh network
vagrant ssh compute
```

随后将用到 Neutron 客户端，请登录到 controller 控制节点（或是安装了 python-neutronclient 包的电脑）。如果使用 Vagrant 创建了该节点，可执行如下命令。

```
vagrant ssh controller
```

确保设置好了如下证书（如果未使用 Vagrant 环境，请调整证书和密钥文件的路径，与环境相匹配）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

有的示例环境中，租户 Neutron 网络可能配置为在 1G 接口上实现隧道；但是在本书的环境中，将使用 eth2 网桥，br-eth2。这个环境也可利用任意数量的网络接口，路由至不同的设备，或者连接至网络中的不同部分。在 OpenStack 网络中，Neutron 会显示管理员可在创建网络时使用新创建的网桥。

以租户管理员的身份，执行如下指令，设置一个新接口网桥，并指定一个 VLAN 类型的 Neutron 网络，实现 192.168.200.0/24 地址上与物理服务器之间的通信。

1. 在 network 和 compute 节点上配置新接口。

```
sudo ifconfig eth4 down
sudo ifconfig eth4 0.0.0.0 up
sudo ip link set eth4 promisc on
```



在运行 Ubuntu 的物理服务器上，可在 /etc/network/interfaces 文件中配置，操作如下。

```
auto eth4
iface eth4 inet manual
    up ip link set $IFACE up
    down ip link set $IFACE down
```


2. 然后, 在 OVS 中新建一个带 eth4 接口的网桥。使用如下命令, 在环境中的所有 compute 和 network 节点执行该步操作。

```
sudo ovs-vsctl add-br br-eth4
sudo ovs-vsctl add-port br-eth4 eth4
```

3. 由于本例中使用了 VLAN 网络, 需要配置 ML2 插件, 使其可发现 VLAN 网络使用的接口。为此, 需编辑/etc/neutron/plugins/ml2/ml2_conf.ini 文件, 插入如下内容。

```
[ml2]
type_drivers = gre,vxlan,vlan
tenant_network_types = vxlan
mechanism_drivers = openvswitch,l2population
```

```
[ml2_type_gre]
tunnel_id_ranges = 1:1000
```

```
[ml2_type_vxlan]
vxlan_group =
vni_ranges = 1:1000
```

```
[vxlan]
enable_vxlan = True
vxlan_group =
local_ip = 10.10.0.202
```

```
[agent]
tunnel_types = vxlan
vxlan_udp_port = 4789
```

```
[ml2_type_vlan]
network_vlan_ranges = physnet4:100:300
```

```
[ovs]
bridge_mappings = physnet4:br-eth4
[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
enable_security_group = True
```

4. 创建 VLAN 网络时可使用新网桥(通过 physnet4 的 bridge_mappings)。本例中, SQL 服务器位于子网络 192.168.200.0/24 上, 其 VLAN ID 为 200。使用如下命令, 创建一个满足这些要求的 Neutron 网络。

```
neutron net-create sqlServerNet \
  --provider:physical_network=physnet4 \
  --provider:network_type=vlan \
  --provider:segmentation_id=200 \
  --shared
```

5. 现在可使用该网络创建子网络。为此, 需要使用 OpenStack 环境中的 192.168.200.0/24 子网络的一部分, 并限制 DHCP 区间以避免与子网络上已有的物理服务器冲突。执行如下

命令。

```
neutron subnet-create sqlServerNet 192.168.200.0/24 \
  --name sqlServerSubnet \
  --allocation-pool start=192.168.200.201,end=192.168.200.240 \
  --gateway 192.168.200.1
```

现在可以使用该网络启动 OpenStack 虚拟机实例了。该网络在子网络 192.168.200.0/24 上被定义为 sqlServerNet，可与同子网络下的物理服务器进行通信。

工作原理

在 compute 和 network 节点上新建了一个接口网桥之后，可在创建 VLAN Neutron 网络时使用。通过以这种方式使用 OpenStack 中的 VLAN 网络，可以在相同子网络和 VLAN 上的 OpenStack 计算云环境与物理服务器之间建立起通信。

为此，要确保新建的 eth4 接口设置为混杂模式 (promiscuous mode)，并且在 OVS 交换机上创建一个名为 br-eth4 的网桥。这个网桥在 /etc/neutron/plugins/ml2/ml2_conf.ini 文件的 [ovs] 部分也有引用，具体如下。

```
[ovs]
bridge_mappings = physnet4:br-eth4
```

这样指定之后，就在网桥与 Neutron 命令行可使用的名称之间建立起了映射，具体见如下 neutron net-create 命令。

```
neutron net-create sqlServerNet \
  --provider:physical_network=physnet4 \
  --provider:network_type=vlan \
  --provider:segmentation_id=200 \
  --shared
```

在上面的命令中，我们还指定了创建的网络类型为 vlan，并指定其 VLAN ID 为 200。

最后，由于是以管理员用户的身份执行的命令，--shared 标志可确保所有租户均可访问该网络。

创建好网络之后，指定与 VLAN 相关联的子网络，限制其 DHCP 区间，避免与该子网络上其他服务器冲突。这步操作通过 neutron subnet-create 命令执行，指定 --allocation-pool 标志。

```
neutron subnet-create sqlServerNet 192.168.200.0/24 \
  --name sqlServerSubnet \
  --allocation-pool start=192.168.200.201,end=192.168.200.240 \
  --gateway 192.168.200.1
```

这里只允许启动 IP 地址区间为 192.168.200.201 至 192.168.200.240 的虚拟机实例。

3.9 配置分布式虚拟路由

OpenStack Juno 版本提供了一个称为分布式虚拟路由（Distributed Virtual Routers, DVR）的特性，与历史版本中的 nova-network 类似。该特性可以让 L3 路由器分布在多个计算节点上，这点与 nova-network 特性相同，因而提供了高可用的路由特性。最终结果是，每个虚拟机实例会连接到位于计算主机的路由器，而不是网络节点的某个中心。这是一个可接受的高可用故障场景：如果某个计算主机出错（导致路由器也出错），也只会影响到运行在那台计算主机上的实例。

图 3-12 显示的是本章使用过的示例环境，但是现在计算主机运行的是 L3 代理。则需要利用 eth3 接口和计算主机的外部网桥 br-ex。在 Legacy L3 路由环境中并不需要该接口，但是对于启用了 DVR 特性的计算主机，在此加入了此前只存在于 network 节点的能力。总体来说，最终的效果类似于在 network 节点上添加了额外的计算和虚拟管理程序功能。

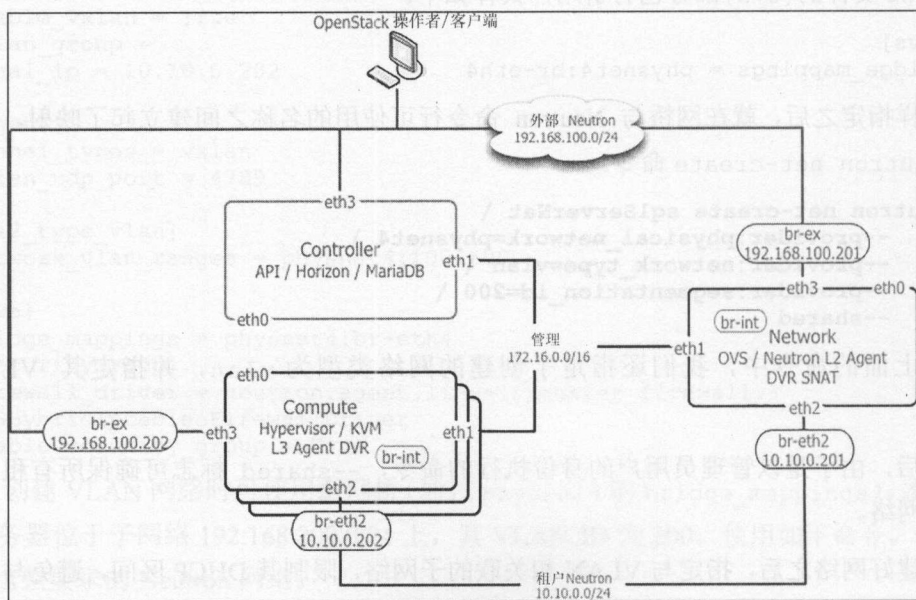


图 3-12

准备工作

为了演示 DVR 特性，要求环境中有一台以上的计算主机。确保有另一台配置好了 Neutron 网络的计算主机。计算主机的完整安装过程请见第 4 章。

如果使用的是 Vagrant 环境，确保环境中运行有另一台计算主机 compute2。编辑 Vagrant 文件，确保 compute2 可用。

确保登陆录了 network、compute、compute2、controller 节点，因为要在这些节点上编辑 DVR 所需的配置文件。如果使用 Vagrant 创建了这些节点，可在不同 shell 下执行如下命令。

```
vagrant ssh controller
vagrant ssh network
vagrant ssh compute
vagrant ssh compute2
```

操作步骤

为了设置好 DVR，需要对环境中的部分 Neutron 配置文件进行略微调整。在接下来的小节中会详细描述。

网络节点

1. 编辑/etc/neutron/neutron.conf 文件，在[DEFAULT]字段下添加如下行设置，指定将使用分布式路由器。

```
router_distributed = True
```

2. 确保/etc/neutron/l3_agent.ini 文件中在[DEFAULT]下有如下内容。

```
[DEFAULT]
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
use_namespaces = True
agent_mode = dvr_snat
external_network_bridge = br-ex
```

3. 编辑/etc/neutron/plugins/ml2/ml2_conf.ini 文件，确保包含如下内容。

```
[ml2]
type_drivers = gre,vxlan,vlan,flat
tenant_network_types = vxlan
mechanism_drivers = openvswitch,l2population
```

```
[ml2_type_gre]
tunnel_id_ranges = 1:1000
```

```
[ml2_type_vxlan]
vni_ranges = 1:1000
```

```
[vxlan]
enable_vxlan = True
local_ip = 10.10.0.201
l2_population = True
```

```
[agent]
tunnel_types = vxlan
l2_population = True
enable_distributed_routing = True
arp_responder = True

[ovs]
local_ip = 10.10.0.201
tunnel_type = vxlan
enable_tunneling = True
l2_population = True
enable_distributed_routing = True
tunnel_bridge = br-tun

[securitygroup]
firewall_driver neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
enable_security_group = True
```

4. 执行如下命令，重启 network 节点上的 Neutron 服务。

```
sudo service neutron-plugin-openvswitch-agent restart
sudo service neutron-l3-agent restart
```

控制节点

接下来，配置 controller 节点上的 Neutron 服务。

1. 编辑/etc/neutron/neutron.conf 文件，在[DEFAULT]字段下添加如下行设置，指定将使用分布式路由器。

```
router_distributed = True
```

2. 使用如下命令，重启 Neutron API 服务，使得改变生效。

```
sudo neutron-server restart
```

计算节点

对于计算节点，需要增加额外的服务，因为将由这些节点，而不是 network 节点来路由大部分的 Neutron 流量。

1. 安装如下软件包。

```
sudo apt-get install neutron-plugin-ml2 \
neutron-plugin-openvswitch-agent \
neutron-l3-agent \
neutron-metadata-agent
```

2. 确保具备外部网络上使用的外部网桥。它将负责路由由外部环境与 SDN 网络之间的流量。

```
sudo ifconfig eth3 down
```

```
sudo ifconfig eth3 0.0.0.0 up
sudo ip link eth3 promisc on
sudo ovs-vsctl add-br br-ex
sudo ovs-vsctl add-port br-ex eth3
```

3. 将此前指派给 eth3 接口的 IP 地址赋给该网桥。

```
sudo ifconfig br-ex 192.168.100.202 netmask 255.255.255.0
```



这个地址位于用来访问 OpenStack 内部实例的网络上。其 IP 区间指定为 192.168.100.0/24，具体见 vagrant 文件描述：

```
network_config.vm.network :hostonly,
"192.168.100.201", :netmask => "255.255.255.0"
```

4. 编辑/etc/neutron/13_agent.ini 文件，添加如下内容。

```
[DEFAULT]
interface_driver = neutron.agent.linux.interface.
OVSSInterfaceDriver
use namespaces = True
agent_mode = dvr
external_network_bridge = br-ex
```

5. 编辑/etc/neutron/metadata_agent.ini 文件，添加如下内容。

```
[DEFAULT]
auth_url = https://192.168.100.200:5000/v2.0
auth_region = RegionOne
admin_tenant_name = service
admin_user = neutron
admin_password = neutron
nova_metadata_ip = 172.16.0.200
auth_insecure = True
metadata_proxy_shared_secret = foo
```

6. 确保/etc/neutron/plugins/ml2/ml2_conf.ini 文件中包含如下内容。

```
[ml2]
type_drivers = gre,vxlan,vlan,flat
tenant_network_types = vxlan
mechanism_drivers = openvswitch,l2population
```

```
[ml2_type_gre]
tunnel_id_ranges = 1:1000
```

```
[ml2_type_vxlan]
vni_ranges = 1:1000
```

```
[vxlan]
enable_vxlan = True
local_ip = 10.10.0.202
l2_population = True
```

```
[agent]
```



```

tunnel_types = vxlan
l2_population = True
enable_distributed_routing = True
arp_responder = True

[ovs]
local_ip = 10.10.0.202
tunnel_type = vxlan
enable_tunneling = True
l2_population = True
enable_distributed_routing = True
tunnel_bridge = br-tun

[securitygroup]
firewall_driver neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
enable_security_group = True

```

7. 使用如下命令，停止并启动 Neutron 的 L3 Agent 服务。

```

sudo stop neutron-l3-agent
sudo start neutron-l3-agent
sudo stop neutron-plugin-openvswitch-agent
sudo start neutron-plugin-openvswitch-agent
sudo stop neutron-metadata-agent
sudo start neutron-metadata-agent

```

工作原理

在安装 Neutron 时做了自定义，添加了 DVR 功能。这将 L2 插件和 L3 代理转移到了计算主机上，这样运行在该主机上的实例将使用 L3 代理作为路由器，而不是运行在 network 节点上的那个中央 L3 路由器。这里执行了类似设置 network 节点时的步骤，不过是在计算主机上复制执行的。这样做是有道理的，因为添加的是此前只在以 Legacy L3 路由模式运行的网络主机上才具备的路由功能。

关键步骤如下。

- ❑ 修改 network 节点的/etc/neutron/l3_agent.ini，加入如下内容。

```

[DEFAULT]
agent_mode = dvr_snat

```

- ❑ 修改 network 节点的/etc/neutron/plugins/ml2/ml2_conf.ini 文件，加入如下内容。

```

[DEFAULT]
mechanism_drivers = openvswitch,l2population

[agent]
tunnel_types = vxlan
l2_population = True

```

```
enable_distributed_routing = True
arp_responder = True
```

```
[ovs]
l2_population = True
enable_distributed_routing = True
```

- ❑ 在控制节点上，编辑/etc/neutron/neutron.conf 文件，设置创建路由器时的默认模式，使其总是为分布式。

```
[DEFAULT]
router_distributed = True
```

- ❑ 在计算节点上，安装如下包。

```
neutron-plugin-m12
neutron-plugin-openvswitch-agent
neutron-l3-agent
neutron-metadata-agent
```

- ❑ 确保启用了外部接口 eth3。然后将其设置为 br-ex 网桥，在外部路由网络中使用。
- ❑ 计算节点上的这些包的配置，与 network 节点上的相同，只有/etc/neutron/13_agent.ini 文件中有一处关键区别（在 network 节点上，这行的设置为 agent_mode = dvr_snat）。

```
[DEFAULT]
agent_mode = dvr
```

3.10 使用分布式虚拟路由器

DVR 模式运行的 Neutron 路由器，创建于计算节点而非网络节点上。这实现了更加分布式的路由，避免了经由网络节点产生的瓶颈。在普通情况下，创建和删除路由器的过程与 Legacy 模式下的相同，但是理解并排查故障时则略有差异。

准备工作

确保有一台适合使用 Neutron 的服务器。如果正在使用配套的 Vagrant 环境，则可以使用 controller 控制节点。该节点上安装了 python-neutronclient 软件包，提供了 neutron 命令行客户端。

如果是通过 Vagrant 创建的该节点，可以执行以下命令。

```
vagrant ssh controller
```

确保设置好了如下证书（如果没有使用 Vagrant 环境，请调整证书和密钥文件的路径，与使用的环境相匹配）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

在本节中，我们将创建一个 DVR 模式运行的路由器，并查看其详情，了解路由器如何与计算主机交互，具体步骤如下。

1. 首先，使用如下命令创建一个路由器。

```
neutron router-create cookbook_router_1
```

输出结果如图 3-13 所示。

Created a new router:	
Field	Value
admin_state_up	True
distributed	True
external_gateway_info	
ha	False
id	93953489-f5a2-42c9-9230-3d8d8c8a2e95
name	cookbook_router_1
routes	
status	ACTIVE
tenant_id	5481d8e9091f49dca2a72b9e223e1f40

图 3-13

图 3-13 显示，新的 distributed 字段被设置为 True。

2. 与前面一样，可将任意网络连接至该路由器。

```
neutron router-interface-add \
    cookbook_router_1 \
    cookbook_subnet_1
```

3. 现在还看不出这个路由器和 Legacy 路由器之间有任何区别。可使用如下命令，定位该路由器。

```
neutron l3-agent-list-hosting-router cookbook_router_1
```

输出结果如图 3-14 所示。

id	host	admin_state_up	alive
0d38b23b-855d-4a11-9765-b97fdb57f2d7	compute	True	:~)

图 3-14

上图中，可以看到该路由器在计算主机中可见，在 network 节点上不可见。

4. 在 Legacy L3 路由模式下，排查路由的名称空间 (Namespace) 时，其形式是 network 节点上的 `qrouter-{netuuid}`。在 DVR 模式下，同样存在这个名称空间，此外还有一个新的 `fip-{extent-uuid}` 名称空间，可用于排查浮动 IP 的指派情况。在计算主机上，执行如下命令。

```
ip netns list
```

输出结果如图 3-15 所示。

```
fip-ca2fc700-b5e2-4c8b-9fa4-6a80f1174360
qrouter-93953489-f5a2-42c9-9230-3d8d8c8a2e95
```

图 3-15

5. 然后可使用该名称空间，测试与指定了浮动 IP 的任意实例之间的连接情况。假设将 192.168.100.11 指派给了计算主机上运行的一个实例。

```
ip netns exec fip-ca2fc700-b5e2-4c8b-9fa4-6a80f1174360 ping
192.168.100.11
```

输出结果如图 3-16 所示。

```
PING 192.168.100.11 (192.168.100.11) 56(84) bytes of data.
64 bytes from 192.168.100.11: icmp_seq=1 ttl=63 time=6.31 ms
64 bytes from 192.168.100.11: icmp_seq=2 ttl=63 time=1.51 ms
64 bytes from 192.168.100.11: icmp_seq=3 ttl=63 time=1.13 ms
64 bytes from 192.168.100.11: icmp_seq=4 ttl=63 time=0.475 ms
^C
--- 192.168.100.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 0.475/2.359/6.313/2.312 ms
```

图 3-16

工作原理

这里讨论了突出分布式路由差异的方法。默认情况下，由于在 `/etc/neutron/neutron.conf` 文件中设置了 `router_distributed = True`，任何正常创建的路由器将会在分布式的计算主机上创建。如需排查路由器问题，可连接到计算主机，并使用创建的名称空间查看。

第 4 章

Nova——OpenStack 计算服务

本章将讲述以下内容：

- ☐ 安装 OpenStack 计算服务控制节点服务
- ☐ 安装 OpenStack 计算软件包
- ☐ 配置数据库服务
- ☐ 配置 OpenStack 计算服务
- ☐ 使用 OpenStack 身份认证服务配置计算
- ☐ 停止与开始 Nova 服务
- ☐ 在 Ubuntu 上安装命令行工具
- ☐ 通过 HTTPS 使用命令行工具
- ☐ 验证 OpenStack 计算服务
- ☐ 使用 OpenStack 计算服务
- ☐ 管理安全组
- ☐ 创建和管理密钥对
- ☐ 启动第一个云实例
- ☐ 修复出错的实例部署
- ☐ 终止实例
- ☐ 使用在线迁移
- ☐ 使用 nova-scheduler
- ☐ 创建实例类型
- ☐ 定义 Host Aggregate
- ☐ 在特定可用区启动实例
- ☐ 在特定计算主机启动实例
- ☐ 从集群移除 Nova 节点

4.1 简介

OpenStack 计算服务，也被称为 Nova 项目，是开源云操作系统 OpenStack 的计算组件。该组件可以在任意数量运行有 OpenStack 计算服务（Compute Service）的主机上运行多个虚拟机实例，通过这种方式可以创建一个高可扩展和高冗余的云环境。这个开源项目力求与硬件和虚拟机管理程序无关。OpenStack 计算服务已经为多个提供计算的大规模云平台提供动力支持，如 Rackspace 的 OpenCloud。

为了帮助读者快速上手，本章将提供相关知识，帮助用户创建一个可运行的云环境。在本章最后，读者将能够使用 OpenStack 工具来创建和访问虚拟机。本章结束后，可完成图 4-1 所示的环境的搭建。

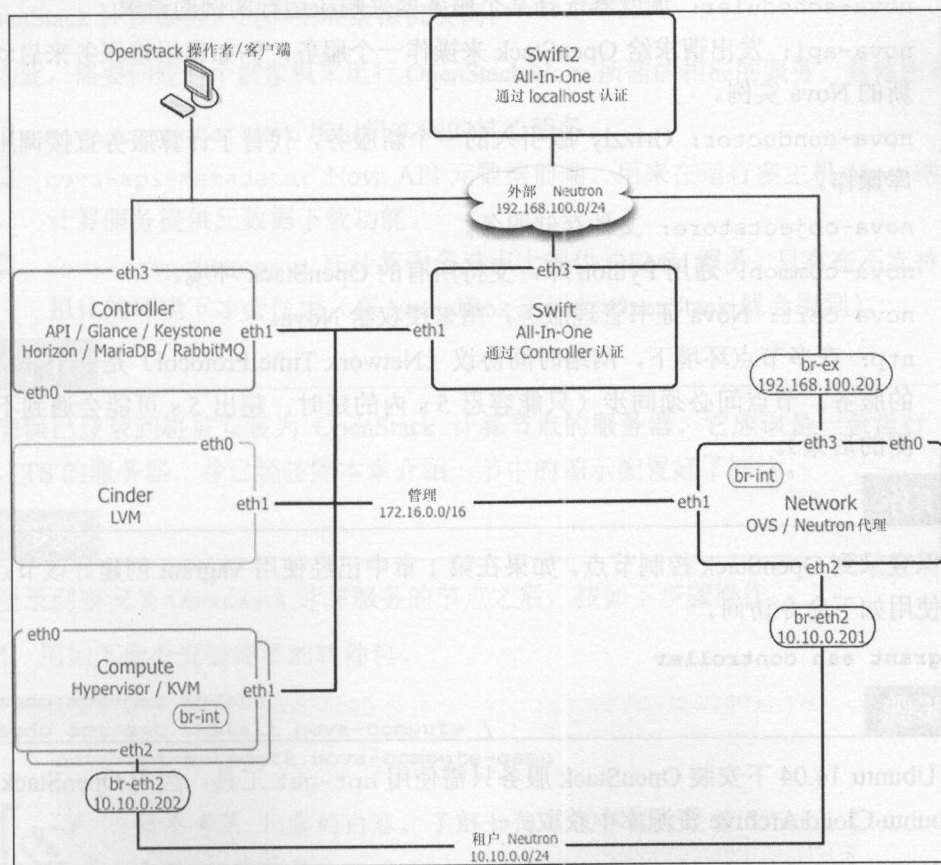


图 4-1



具体来说，搭建的是图中 **Compute** 部分。

4.2 安装 OpenStack 计算控制节点服务

在创建一个运行 OpenStack 计算服务的服务器来运行实例之前，需要将一些最终的服务与 OpenStack 身份认证服务和镜像服务一起提前安装在控制节点上。把控制服务和计算服务分离开，可以为 OpenStack 提供水平扩展的能力。

为此，首先安装一些服务到控制节点上，这个节点也就是之前第 1 章、第 2 章和第 3 章中安装过 Keystone、Glance 和 Neutron 服务的节点。具体服务如下。

- ❑ nova-scheduler: 调度器选择某个服务器来响应运行实例的请求。
- ❑ nova-api: 发出请求给 OpenStack 来操作一个服务，例如调用该服务来启动一个新的 Nova 实例。
- ❑ nova-conductor: Grizzly 版引入的一个新服务，代替了计算服务直接调用数据库操作。
- ❑ nova-objectstore: 文件存储服务。
- ❑ nova-common: 通用 Python 库，支持所有的 OpenStack 环境。
- ❑ nova-cert: Nova 证书管理服务，用来授权给 Nova。
- ❑ ntp: 在多节点环境下，网络时间协议（Network Time Protocol）是一个非常重要的服务，节点间必须同步（只能容忍 5 s 内的延时，超出 5 s 可能会遇到不能预测的后果）。

准备工作

确保登录到 OpenStack 控制节点。如果在第 1 章中已经使用 Vagrant 创建好该节点，可以直接使用如下命令访问。

```
vagrant ssh controller
```

操作步骤

在 Ubuntu 14.04 下安装 OpenStack 服务只需使用 apt-get 工具，因为 OpenStack 包可以从 Ubuntu Cloud Archive 资源库中获取。

可使用如下命令安装所需的软件包。

```
sudo apt-get update
sudo apt-get install nova-api \
    nova-conductor nova-scheduler nova-objectstore
```

工作原理

从 Ubuntu Cloud Archive 软件包资源库中安装 OpenStack 计算控制节点所需的软件包非常轻松，并且可以及时获取最新的 OpenStack 功能。同时也添加了一份稳定的保证，确保不会偏离或远离主干分支的升级路线。

4.3 安装 OpenStack 计算软件包

有了一个可以运行 OpenStack 计算服务的虚拟机后，便可以通过安装相应的软件包运行 OpenStack 计算服务，进而生成虚拟机实例。

为此，需要创建一个虚拟机来运行 OpenStack Nova 所需的相应的服务。具体服务如下。

- ❑ nova-compute: 运行虚拟机实例的核心服务。
- ❑ nova-api-metadata: Nova API 元数据前端。用来在运行多主机 Nova 网络中为计算服务提供元数据下载功能。
- ❑ nova-compute-qemu: 在计算服务节点上提供 QEMU 服务。只有在不支持硬件虚拟化的环境下才会使用（在 VirtualBox 下运行 OpenStack 就会用到）。

准备工作

确保已登录到将被安装为 OpenStack 计算节点的服务器。它应该是一台运行 Ubuntu 14.04 LTS 的服务器，并已经按照本章介绍一节中的图示配置好了网络。

操作步骤

登录到要安装 OpenStack 计算服务的节点之后，按如下步骤操作。

1. 用如下命令安装需要的软件包。

```
sudo apt-get update
sudo apt-get install nova-compute \
    nova-api-metadata nova-compute-qemu
```



请参考第 1 章的内容，了解如何在服务器上设置 Ubuntu Cloud Archive 资源库。

安装完成后，需要安装和配置 ntp。

```
sudo apt-get install ntp
```

2. NTP 在任何多节点环境中都是非常重要的，而且在 OpenStack 环境中需要用它来与服务器同步时间。为此，编辑/etc/ntp.conf 文件，加入以下内容。

```
# Replace ntp.ubuntu.com with an NTP server on your network
server ntp.ubuntu.com
server 127.127.1.0
fudge 127.127.1.0 stratum 10
```

3. 正确配置好 NTP 后，重新启动服务使变更生效。

```
sudo service ntp restart
```

工作原理

从 Ubuntu Cloud Archive 软件包资源库中安装 OpenStack 计算服务非常简单易懂，通过它可以方便地在 Ubuntu 服务器上搭建最新的 OpenStack。使用该方法还能为 OpenStack 的稳定运行和升级带来更高的可靠性，因为其始终可以和主归档文件保持一致。

更多参考

安装 OpenStack 还有多种方式，从源代码构建到用包安装都可以，但以上方法是最简单和最常用的。另外，还有其他 OpenStack 发布版本可以获取。使用 Ubuntu Cloud Archive 还可以在 Ubuntu 14.04 LTS 平台使用不同的 OpenStack 版本。

使用其他版本

当参与 OpenStack 开发或调试时，或者所需要的功能不在当前发布版本时，不选择稳定版本也是可以的。要想使用其他发布版本，需要添加不同的 **Personal Package Archives (PPA)** 到系统中。访问 <http://wiki.openstack.org/ppas>，查看 OpenStack PPA。如要使用它们，先安装一个工具就能够轻松地将 PPA 添加到系统中。

```
sudo apt-get update
sudo apt-get install software-properties-common python-software-properties
```

如需使用特定版本的 PPA，如 Kilo，可以使用以下命令。

```
sudo add-apt-repository cloud-archive:kilo
```


4.4 配置数据库服务

OpenStack 支持多种数据库，如内置的 SQLite 数据库（默认）、MySQL 数据库和 Postgres 数据库。SQLite 只用于测试，不支持也不应该用于生产环境，选择 MySQL 还是 Postgres，取决于数据库管理员的经验。本书前面有提到，这里将使用 MariaDB。

准备工作

由于要将 OpenStack 控制服务配置为使用 MariaDB 作为数据库后端，所以要在配置 OpenStack 计算服务之前安装数据库。



有关设置 MariaDB 的指令，可参考配套网站：<http://bit.ly/OpenStackCookbookPreReqs>。如需设置 MariaDB 获取高可用性，可参考第 11 章 11.2 节。

如果还没有登录到 OpenStack 控制节点，则使用 Vagrant 命令 ssh 登录到该服务器。

操作步骤

要使用 OpenStack 计算服务（Nova），首先需要保证有一个名为 nova 的后台数据库。为此，需要在运行数据库的控制节点上执行以下命令。

1. 运行 MySQL，配置一个名为 nova 的数据库用户，并赋予 OpenStack 计算服务所需要的权限。

```
MYSQL_ROOT_PASS=openstack
```

```
mysql -uroot -p$MYSQL_ROOT_PASS -e 'CREATE DATABASE nova;'
```

```
MYSQL_NOVA_PASS=openstack
```

```
mysql -uroot -p${MYSQL_ROOT_PASSWORD} \
-e "GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' IDENTIFIED BY
'${MYSQL_NOVA_PASSWORD}';"
mysql -uroot -p${MYSQL_ROOT_PASSWORD} \
-e "GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost'
IDENTIFIED BY '${MYSQL_NOVA_PASSWORD}';"
```

2. 现在，需在 /etc/nova/nova.conf 文件中引用这个 MySQL 服务器。添加 sql_connection 标志就可以使用 MySQL 了。

```
sql_connection=mysql://nova:openstack@192.168.100.200/nova
```

工作原理

MySQL 对于 OpenStack 来说是个基础的服务，很多服务都将依赖于它。正确配置 MySQL 才能确保服务器顺利运行。添加了一个名为 nova 的数据库后，它最终会被 OpenStack 计算服务的相关表和数据填充。同时赋予 nova 数据库所需的用户权限以便能够访问它。最后配置 OpenStack。

在 OpenStack 计算服务安装过程中指定配置详情以使用 nova 数据库。

延伸阅读

□ 参见 11.2 节。

4.5 配置 OpenStack 计算服务

/etc/nova/nova.conf 是一个重要的配置文件，本书将会多次提到。该文件描述了每个 OpenStack 计算服务如何运行，以及需要链接到哪些服务才能呈现给终端用户。当环境不断扩展时，该文件将在节点间被复制。



同一个 /etc/nova/nova.conf 文件可以被环境中所有的 OpenStack 计算服务节点使用。一旦创建好，就可以复制给环境中的所有其他节点使用。

准备工作

首先，需要在控制主机和计算主机上配置 /etc/nova/nova.conf 文件。

如果使用的是本书提供的 Vagrant 环境，可以执行如下命令登录到 OpenStack 控制主机和计算主机。

```
vagrant ssh controller
vagrant ssh compute-01
```

操作步骤

为了运行沙盒环境，需要配置 OpenStack 计算服务，使之能够被底层主机访问。还要配置 API 服务监听的外网接口，将其他服务配置为正确的端口，这样客户端工具才能与这些服务通信。本节将逐行介绍沙盒环境使用的整个 nova.conf 文件。

1. 首先，为 /etc/nova/nova.conf 文件添加以下内容。

```
[DEFAULT]
```

```

dhcpbridge_flagfile=/etc/nova/nova.conf
dhcpbridge=/usr/bin/nova-dhcpbridge
logdir=/var/log/nova
state_path=/var/lib/nova
lock_path=/var/lock/nova
root_wrap_config=/etc/nova/rootwrap.conf
verbose=True

use_syslog = True
syslog_log_facility = LOG_LOCAL0

api_paste_config=/etc/nova/api-paste.ini
enabled_apis=ec2,osapi_compute,metadata

# Libvirt and Virtualization
libvirt_use_virtio_for_bridges=True
connection_type=libvirt
libvirt_type=qemu

# Messaging
rabbit_host=192.168.100.200

# EC2 API Flags
ec2_host=192.168.100.200
ec2_dmz_host=192.168.100.200
ec2_private_dns_show_ip=True

# Network settings
network_api_class=nova.network.neutronv2.api.API
neutron_url=http://192.168.100.200:9696
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_admin_username=neutron
neutron_admin_password=neutron
neutron_admin_auth_url=https://192.168.100.200:5000/v2.0
libvirt_vif_driver=nova.virt.libvirt.vif.
LibvirtHybridOVSBridgeDriver
linuxnet_interface_driver=nova.network.linux_net.
LinuxOVSInterfaceDriver
#firewall_driver=nova.virt.libvirt.firewall.IptablesFirewallDriver
security_group_api=neutron
firewall_driver=nova.virt.firewall.NoopFirewallDriver
neutron_ca_certificates_file=/etc/ssl/certs/ca.pem

service_neutron_metadata_proxy=true
neutron_metadata_proxy_shared_secret=foo

#Metadata
#metadata_host = 192.168.100.200

```



```

#metadata_listen = 192.168.100.200
#metadata_listen_port = 8775

# Cinder #
volume_driver=nova.volume.driver.ISCSIDriver
volume_api_class=nova.volume.cinder.API
iscsi_helper=tgtadm
iscsi_ip_address=172.16.0.200

# Images
image_service=nova.image.glance.GlanceImageService
glance_api_servers=192.168.100.200:9292

# Scheduler
scheduler_default_filters=AllHostsFilter

# Auth
auth_strategy=keystone
keystone_ec2_url=https://192.168.100.200:5000/v2.0/ec2tokens

# NoVNC
novnc_enabled=true
novncproxy_host=192.168.100.200
novncproxy_base_url=http://192.168.100.200:6080/vnc_auto.html
novncproxy_port=6080

xvpvncproxy_port=6081
xvpvncproxy_host=192.168.100.200
xvpvncproxy_base_url=http://192.168.100.200:6081/console

vncserver_proxycient_address=192.168.100.200
vncserver_listen=0.0.0.0

# Database
[database]
sql_connection=mysql://nova:openstack@192.168.100.200/nova

[keystone_authtoken]
auth_host = 192.168.100.200
auth_port = 35357
auth_protocol = https
admin_tenant_name = service
admin_user = nova
admin_password = nova
insecure = True

```

2. 重复第1步，在计算主机上创建/etc/nova/nova.conf文件。

3. 回到控制主机，通过如下命令确保数据库拥有正确的表结构，并且填入正确的初始

数据信息。

```
sudo nova-manage db sync
```



该命令执行成功是没有输出的。

工作原理

/etc/nova/nova.conf 文件在 OpenStack 环境中非常重要，所有的计算节点和控制节点都会使用同样的文件。它只需创建一次，并确保所有的节点上都有该文件，下面是 /etc/nova/nova.conf 配置文件里的标志。

- ❑ dhcpbridge_flatfile=: 指定了 dhcpbridge 服务的配置 (flag) 文件位置。
- ❑ dhcpbridge=: 指定了 dhcpbridge 服务的位置。
- ❑ logdir=/var/log/nova: 记录所有服务的日志，该区域只能由 root 用户写。
- ❑ state_path=/var/lib/nova: 主机上 Nova 用于维护运行着的服务状态的位置。
- ❑ lock_path=/var/lock/nova: Nova 写锁文件位置。
- ❑ root_wrap_config=/etc/nova/rootwrap.conf: 指定一个帮助脚本，允许 OpenStack 计算服务获取 root 特权。
- ❑ verbose: 设置是否需要更多的信息显示出来。
- ❑ use_syslog: 将日志发送给 syslog 日志消息来源 (facility)。
- ❑ syslog_log_facility: 指定使用哪个日志消息来源。这里为所有服务设置同一个日志消息来源，LOG_LOCAL0。生产环境上，推荐为不同服务使用不同的日志消息来源。
- ❑ api_paste_config: 解析文件的位置，包括 nova-api 服务的 paste.deploy 配置。
- ❑ enabled_apis: 指定默认使用的 API。
- ❑ connection_type=libvirt: 指定用来使用 libvirt 的连接。
- ❑ libvirt_use_virtio_for_bridges: 桥接使用的 virtio 驱动。
- ❑ libvirt_type=qemu: 设置虚拟化模式。qemu 是软件虚拟化，底层需要运行 VirtualBox。其他选项包括 kvm 和 xen。
- ❑ sql_connection=mysql://nova:openstack@192.168.100.200/nova: 前面章节创建的 SQL 链接。它表示“用户:密码@主机地址/数据库”名称(本例中是 nova)。
- ❑ rabbit_host=192.168.100.200: 通知 OpenStack 服务到哪里去查找 rabbitmq

消息队列服务。

- ❑ `ec2_host=192.168.100.200`: 表示 nova-api 服务的外部 IP 地址。
- ❑ `ec2_dmz_host=192.168.100.200`: 表示 nova-api 服务的内部 IP 地址。
- ❑ `ec2_private_dns_show_ip`: 如果设为 `true`, 返回 IP 地址, 否则返回私有主机名。
- ❑ `network_api_class`: 设置使用的网络 API 类的全称。
- ❑ `neutron_url`: 设置 Neutron (网络) 服务的 API。
- ❑ `neutron_auth_strategy`: 设置验证策略, 这里使用 Keystone。
- ❑ `neutron_admin_tenant_name`: 设置验证 Neutron 服务时使用的租户。
- ❑ `neutron_admin_username`: 设置验证 Neutron 服务时使用的用户名。
- ❑ `neutron_admin_password`: 设置验证 Neutron 服务时使用的密码。
- ❑ `neutron_admin_auth_url`: 设置验证 Neutron 服务时使用的端点。
- ❑ `libvirt_vif_driver`: 设置 Nova 安全过滤使用 VIF 插件。
- ❑ `linuxnet_interface_driver`: 设置用来创建以太网设备的驱动。
- ❑ `security-group_api`: 设置安全 API 的分类名。
- ❑ `firewall_driver`: 设置防火墙驱动。
- ❑ `neutron_ca_certificates_file`: 设置 SSL 验证时使用的证书文件。
- ❑ `service_neutron_metadata_proxy`: 表示用作验证元数据代理的计算节点。
- ❑ `neutron_metadata_proxy_shared_secret`: 设置元数据代理使用的密文。
- ❑ `volume_driver`: 设置卷驱动器类的全称。
- ❑ `volume_api_class`: 设置使用的卷 API 的全称。
- ❑ `iscsi_helper`: 指定使用 `tgtadm` 守护程序作为 iSCSI 的用户工具。
- ❑ `iscsi_ip_address`: 设置 iSCSI 的 IP 地址。
- ❑ `image_service`: 指定使用 Glance 服务管理镜像。
- ❑ `glance_api_servers`: 指定运行 Glance 镜像服务的服务器。
- ❑ `scheduler_default_filters`: 指定调度器可向所有计算主机发送请求。
- ❑ `auth_strategy`: 指定使用 Keystone 验证所有服务。
- ❑ `keystone_ec2_url`: 设置指定 Keystone 服务器 ec2 的 URL。
- ❑ `novnc_enabled`: 为计算实例启用 noVNC 客户端, 通过 Web 浏览器提供 VNC。
- ❑ `novncproxy_host`: 指定 noVNC 代理 IP。
- ❑ `novncproxy_base_url`: 指定 noVNC 代理的基础 URL。
- ❑ `novncproxy_port`: 指定 noVNC 代理的端口。
- ❑ `xvpvncproxy_port`: 指定 Nova 服务使用的 XVP VNC 控制台代理端口。

- ❑ `xvpvncproxy_host`: 指定 Nova 服务使用的 XVP VNC 的 IP 地址。
- ❑ `xvpvncproxy_base_url`: 指定 Nova 服务使用的 XVP VNC 控制台的 URL。
- ❑ `vncserver_proxyclient_address`: 指定 VNC 服务器代理客户端地址。
- ❑ `vncserver_listen`: 表示 VNC 服务器监听的端口。
- ❑ `auth_host`: 设置 Keystone 服务的 IP 地址。
- ❑ `auth_port`: 设置 Keystone 服务的端口。
- ❑ `auth_protocol`: 设置验证协议, 这里使用的是 HTTPS。
- ❑ `admin_tenant_name`: 设置验证时的租户名称。
- ❑ `admin_user`: 设置计算服务验证 Keystone 服务时使用的用户名。
- ❑ `admin_password`: 设置验证 Keystone 服务时使用的密码。

修改上述配置选项之后, 重启 Nova 服务。具体操作将在 4.7 节中介绍。

更多参考

还有很多选项用于配置 OpenStack 计算服务。接下来的章节将会继续介绍这些选项, 毕竟 `nova.conf` 文件是大部分 OpenStack 计算服务的基础。

延伸阅读

可以在 OpenStack 网站 (<http://docs.openstack.org/juno/config-reference/content/list-of-computeconfig-options.html>) 上查看每个配置项的用途描述。

4.6 使用 OpenStack 身份认证服务配置计算服务

安装和配置好 OpenStack 身份认证服务 (Keystone) 之后, 需要告知 OpenStack 计算服务 (Nova) 如何使用身份认证服务来认证用户和服务了。



以下步骤将会在环境中所有的控制节点和计算节点上重复操作。

准备工作

在开始之前, 确保已经登录到 OpenStack 计算节点和控制主机。如果是使用 Vagrant 创建的 OpenStack 节点, 可以在不同的 shell 中执行以下命令。

```
vagrant ssh controller
vagrant ssh compute-01
```

操作步骤

执行如下几个简单步骤，即可在 OpenStack 计算沙盒环境中配置认证机制。

1. 首先，如果该计算节点是一个单机（standalone）的计算节点，需要确保在 OpenStack 计算主机上已安装好必需的 python-keystone 软件包，执行如下命令。

```
sudo apt-get update
sudo apt-get install python-keystone
```

2. 为了将 OpenStack 计算服务配置为可以使用 OpenStack 身份认证服务的模式，第一步要在 /etc/nova/api-paste.ini 文件的 [DEFAULT] 和 [keystone_authtoken] 中添加第 1 章 1.8 节中我们为 Nova 服务用户创建的详细信息，代码如下。

```
[DEFAULT]
api_paste_config=/etc/nova/api-paste.ini
auth_strategy=keystone
keystone_ec2_url=https://192.168.100.200:5000/v2.0/ec2tokens
[keystone_authtoken]
admin_tenant_name = service
admin_user = nova
admin_password = nova
identity_uri = https://192.168.100.200:35357/
insecure = True
```

3. nova.conf 文件配置好之后，编辑 /etc/nova/api-pase.ini，在 [filter:keystonecontext] 和 [filter:authtoken] 部分下添加如下行，设置 keystone 为认证机制。

```
[filter:keystonecontext]
paste.filter_factory = nova.api.auth:NovaKeystoneContext.factory

[filter:authtoken]
paste.filter_factory = keystonemiddleware.auth_token:filter_factory
```

4. OpenStack 身份认证服务运行起来后，重启 OpenStack 计算服务来使认证机制的修改生效，命令如下。

```
ls /etc/init/nova-* | cut -d '/' -f4 | cut -d '.' -f1 | while read
S; do sudo stop $S; sudo start $S; done
```

工作原理

为了将所有 OpenStack 计算服务配置为在此环境中可以使用 OpenStack 的身份认证服务，先要编辑 /etc/nova/nova.conf 文件，加入证书和 Keystone 设置。

然后配置 /etc/nova/api-past.ini 文件，在 [filter:keystonecontext] 和 [filter:authtoken] 部分里添加 keystone factory 的详细信息。

4.7 停止和启动 Nova 服务

配置好 OpenStack 计算服务的安装环境后，便可以启动 OpenStack 虚拟机（控制节点和计算节点）来运行服务了，然后准备好启动私有云实例。

准备工作

将 ssh 登录到控制和计算节点虚拟机。如果是使用 Vagrant 创建的这些节点，在不同的 shell 中分别使用以下命令。

```
vagrant ssh controller
vagrant ssh compute-01
```

这确保能够访问虚拟机，因为需要从个人电脑连接过去启动实例。沙盒环境中运行的 OpenStack 服务具体如下。

在控制节点上：

- ☐ nova-api;
- ☐ nova-objectstore;
- ☐ nova-scheduler;
- ☐ nova-conductor;
- ☐ nova-cert;
- ☐ nova-novncproxy;
- ☐ nova-consoleauth.

在计算节点上：

- ☐ nova-compute;
- ☐ nova-api-metadata;
- ☐ nova-novncproxy;
- ☐ libvirt-bin.

操作步骤

执行以下步骤，停止正在运行的 OpenStack 计算服务。


1. 在安装过程中，OpenStack 计算服务默认启动。因此，第一步需要通过以下命令停止服务。

在控制节点上执行如下命令。

```
sudo stop nova-api
sudo stop nova-scheduler
sudo stop nova-objectstore
sudo stop nova-conductor
sudo stop nova-cert
sudo stop nova-novncproxy
sudo stop nova-consoleauth
```

在计算节点上执行如下命令。

```
sudo stop nova-compute
sudo stop nova-api-metadata
sudo stop nova-novncproxy
```

 可以使用以下命令，停止所有 OpenStack 计算服务。

```
ls /etc/init/nova-* | cut -d '/' -f4 | cut -d '.' -f1
| while read S; do sudo stop $$; done
```

2. 还有 libvirt 服务同样需要停止。

```
sudo stop libvirt-bin
```

执行以下步骤，启动 OpenStack 计算服务。


3. 启动 OpenStack 计算服务的过程和停止它们的过程类似。

在控制节点上执行如下命令。

```
sudo start nova-api
sudo start nova-scheduler
sudo start nova-objectstore
sudo start nova-conductor
sudo start nova-cert
sudo start nova-novncproxy
sudo start nova-consoleauth
```

在计算节点上执行如下命令。

```
sudo start nova-compute
sudo start nova-network
sudo start nova-api-metadata
sudo start nova-novncproxy
```

 使用以下命令可以一次性启动所有 OpenStack 的计算服务。

```
ls /etc/init/nova-* | cut -d '/' -f4 | cut -d
'.' -f1 | while read S; do sudo start $$; done
```

4. 安装的 libvirt 服务现在也要启动。

```
sudo start libvirt-bin
```

工作原理

在 Ubuntu 下，可使用 upstart 脚本来停止和启动 OpenStack 计算服务。使用这种方式，可以用 start 和 stop 命令并加上服务的名字来简单地控制服务的运行。

4.8 在 Ubuntu 上安装命令行工具

可以使用 Nova 客户端从命令行管理 OpenStack 计算。Nova 客户端工具使用 OpenStack 计算 API 和 OS-API。如想体会云环境的灵活性和强大，学会这些工具是非常重要的，因为这有助于创建强大的脚本来管理云。

准备工作

将这些工具安装在运行 Ubuntu 的主机上。这是获取 Nova 客户端软件包来管理云环境的最容易的方法。

操作步骤

Nova 客户端软件包可以很方便地从 Ubuntu 资源库获取。如果主机运行的不是 Ubuntu，在 OpenStack 计算虚拟机之外再创建一个 Ubuntu 虚拟机是使用这些工具的最简单的方式。

在 Ubuntu 机器上，输入以下命令。

```
sudo apt-get update
sudo apt-get install python-novaclient
```

工作原理

在 Ubuntu 上使用 Nova 客户端是管理 OpenStack 云环境的标准方式。安装非常简单，只需通过标准的 Ubuntu 软件包安装方式即可完成。

延伸阅读

更多信息查看 <http://bit.ly/OpenStackCookbookClientInstall>。

4.9 通过 HTTPS 使用命令行工具

由于 OpenStack 身份认证端点配置为使用 HTTPS，在使用命令行工具操作 OpenStack

计算服务时需要指定 SSL 证书。

准备工作

首先将这些工具安装在正在运行 Ubuntu 的主机上。这是获取 Nova 客户端软件包来管理云环境的最容易的方法。如果使用的是 Vagrant 实验环境，则已经安装并配置了自签名的证书。但是建议在生产环境中使用受信任的证书机构（Certificate Authority）颁发的证书。

操作步骤

可以很方便地从 Ubuntu 资源库中安装 Nova 客户端软件包。已经安装好了 SSL 证书，并配置 Keystone 服务使用证书进行验证。

1. 以普通用户身份在 Ubuntu 机器上输入如下命令。

```
sudo apt-get update
sudo apt-get install python-novaclient
```

2. 安装命令行工具之后，设置环境变量。这里需要使用 OpenStack 集群的 SSL 证书。

如果使用的不是 Vagrant 环境，请根据情况调整证书和密钥文件的路径。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```



注意，这里设置 OS_KEY 环境变量为私钥文件的路径。OS_CACERT 变量指向验证 TLS (https) 服务器证书时使用的证书文件路径。

3. 排查证书或连接问题时，可使用 --insecure 标志绕过 SSL 验证。在 nova 命令行客户端使用该标志时，不会向任何证书机构验证服务器的证书。



注意，--insecure 标志在排查连接问题时非常有用。它还可以绕过所有的证书验证——甚至可能都没有启用证书验证。

工作原理

在 Ubuntu 上使用 Nova 客户端，是管理 OpenStack 环境的标准方式。但是，如果身份验证端点被配置为使用 HTTPS，则需要将命令行工具指向系统上安装的证书。添加指向证书的环境变量可以自动进行验证。

4.10 检查 OpenStack 计算服务

至此,已经安装好 OpenStack 计算,接下来需要验证一下配置是否符合预期。OpenStack 计算提供了许多工具来检测各部分的配置。同时,这里还会用到常用的系统命令来验证那些支撑 OpenStack 计算的底层服务是否也按预期运行。

准备工作

登录到 OpenStack 控制节点。如果这个节点是使用 Vagrant 创建的,可以使用如下命令访问。

```
vagrant ssh controller
```

操作步骤

要验证 OpenStack 计算服务是否正在运行,可以调用 nova-manage 工具查询关于环境的信息,如下所示。

1. 检查 OpenStack 计算主机是否运行正常:

```
sudo nova-manage service list
```

会看到图 4-2 所示的输出,其中的:-) 图标表示一切正常。

Binary	Host	Zone	Status	State	Updated_At
nova-scheduler	controller	internal	enabled	:-)	2015-01-28 04:25:16
nova-consoleauth	controller	internal	enabled	:-)	2015-01-28 04:25:19
nova-cert	controller	internal	enabled	:-)	2015-01-28 04:25:19
nova-conductor	controller	internal	enabled	:-)	2015-01-28 04:25:15
nova-consoleauth	compute-01	internal	enabled	:-)	2015-01-28 04:25:16
nova-compute	compute-01	nova	enabled	:-)	2015-01-28 04:25:20
nova-consoleauth	compute-02	internal	enabled	:-)	2015-01-28 04:25:18
nova-compute	compute-02	nova	enabled	:-)	2015-01-28 04:25:15

图 4-2

2. 如果在应该出现:-) 的地方看到了 xxx,就说明遇到了问题,如图 4-3 所示。

Binary	Host	Zone	Status	State	Updated_At
nova-scheduler	controller	internal	enabled	:-)	2015-01-28 05:26:38
nova-consoleauth	controller	internal	enabled	:-)	2015-01-28 05:26:42
nova-cert	controller	internal	enabled	:-)	2015-01-28 05:26:41
nova-conductor	controller	internal	enabled	:-)	2015-01-28 05:26:46
nova-consoleauth	compute-01	internal	enabled	:-)	2015-01-28 05:26:38
nova-compute	compute-01	nova	enabled	XXX	2015-01-28 05:25:41
nova-consoleauth	compute-02	internal	enabled	:-)	2015-01-28 05:26:40
nova-compute	compute-02	nova	enabled	:-)	2015-01-28 05:26:46

图 4-3

如果看到 xxx,一般可以在 /var/log/nova/ 目录下的日志里找到答案。



若是看到某个服务的 XXX 和:-) 图标交错出现, 应先检查系统时钟是否同步。

3. 没有专门检查 Glance 的工具, 所以需要使用一些系统命令。

```
ps -ef | grep glance
netstat -ant | grep 9292.*LISTEN
```

这些命令会返回 Glance 的进程信息, 显示它是否正在运行并且默认端口是 9292, 这个端口应该已经在服务器上打开且处于 LISTEN 模式。

4. 其他需要检查的服务还有以下几个。

❑ rabbitmq:

```
sudo rabbitmqctl status
```

若一切运行正常, 会看到 rabbitmqctl 有类似图 4-4 所示的输出。

```
Status of node rabbit@controller ...

[{pid,26086},
 {running_applications,[{rabbit,"RabbitMQ","3.2.4"},
                        {os_mon,"CPD CXC 138 46","2.2.14"},
                        {mnesia,"MNESIA CXC 138 12","4.11"},
                        {xmerl,"XML parser","1.3.5"},
                        {sasl,"SASL CXC 138 11","2.3.4"},
                        {stdlib,"ERTS CXC 138 10","1.19.4"},
                        {kernel,"ERTS CXC 138 10","2.16.4"}]},
 {os,{unix,linux}},
 {erlang_version,"Erlang R16B03 (erts-5.10.4) [source] [64-bit] [smp:2:2] [async-threads:30] [kernel-
poll:true]\n"},
 {memory,[{total,128095000},
          {connection_procs,3760040},
          {queue_procs,1981200},
          {plugins,0},
          {other_proc,13575624},
          {mnesia,355808},
          {mgmt_db,0},
          {msg_index,84024},
          {other_ets,918776},
          {binary,86007184},
          {code,16522377},
          {atom,594537},
          {other_system,4295430}}],
 {vm_memory_high_watermark,0.4},
 {vm_memory_limit,1303538892},
 {disk_free_limit,50000000},
 {disk_free,34748444672},
 {file_descriptors,[{total_limit,924},
                   {total_used,88},
                   {sockets_limit,829},
                   {sockets_used,86}]},
 {processes,[{limit,1048576},{used,1013}]},
 {run_queue,0},
 {uptime,90276}]
...done.
```

图 4-4

❑ ntp (Network Time Protocol, 保持节点同步):

```
ntpq -p
```

该命令会输出与 NTP 服务器通信相关的信息，如图 4-5 所示。

remote	refid	st	t	when	poll	reach	delay	offset	jitter
*yurizoku.tk	209.51.161.238	2	u	104	1024	377	46.426	2.183	1.889
-propjet.latt.ne	187.253.153.32	2	u	56	1024	377	47.226	5.163	1.253
+juniperberry.ca	192.93.2.20	2	u	395	1024	377	113.354	2.171	1.618

图 4-5

❑ MySQL 数据库服务器：

```
MYSQL_ROOT_PASS=openstack
mysqladmin -uroot -p{$MYSQL_ROOT_PASS}status
```

如果 MySQL 正在运行，则会返回一些关于 MySQL 的统计数据，如图 4-6 所示。

```
Uptime: 91271 Threads: 21 Questions: 762166 Slow queries: 0 Opens: 1745 Flush tables: 1 Open tables:
332 Queries per second avg: 8.350
```

图 4-6

工作原理

这里使用了一些基本的命令与 OpenStack 计算服务和其他服务通信，查看它们是否都正常运行。这些基础的故障排除可以保证系统正在如预期的那样运行。

- ❑ `sudo nova-manage service list`：列出各种 Nova 服务和对应的状态。
- ❑ `ps -ef | grep glance`：列出正在进行的 Glance 服务。
- ❑ `netstat -ant | grep 9292.*LISTEN`：允许检查 Glance 守护程序是否在网络上进行监听。
- ❑ `sudo rabbitmqctl status`：允许验证 rabbitMQ 服务是否在运行。
- ❑ `ntpq -p`：确认 NTP 是否运行正常，是否连接到了配置的远程服务器。
- ❑ `mysqladmin -uroot -p{$MYSQL_ROOT_PASS} status`：返回 MySQL 进程的基本信息。

4.11 使用 OpenStack 计算服务

OpenStack 身份认证服务是所有 OpenStack 服务的基石。由于 OpenStack 镜像服务也同样配置为使用 OpenStack 认证服务，现在的 OpenStack 计算环境已经是可用的了。

准备工作

首先，登录到 Ubuntu 客户端，并确保 Nova 客户端是可用的。如果不可用，则可以像

下面这样安装。

```
sudo apt-get update
sudo apt-get python-novaclient
```

操作步骤

要使用 OpenStack 认证服务作为 OpenStack 环境中的认证机制，需要设置相应环境变量。下面演示如何进行设置。

1. Nova 客户端安装完成后，还需要配置相应环境变量才能使用它。具体步骤如下。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=http://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

将这几行命令加入到 home 目录的 novarc 文件。这样每次只需执行如下命令即可简单地导入这些认证信息。

novarc



注意，如果在 shell 中已经设置了用户证书环境变量，并且其中包含 SERVICE_TOKEN 和 SERVICE_ENDPOINT 变量的设置，这将覆盖上一步设置的用户证书信息。在继续之前，删除 SERVICE_TOKEN 和 SERVICE_ENDPOINT 变量。

2. 要访问创建的 Linux 实例，必须先创建一个密钥对（keypair）来得到访问权限。密钥对是一对 SSH 私钥和公钥，有了它才能访问一个虚拟机。用户只要保证密钥的安全，公钥可以分发给任何人或任何机器而无须担心安全问题。因为只有私钥匹配才能获得虚拟机的授权。云主机实例就是依靠密钥对来控制访问的。接下来，使用 Nova 客户端创建一个密钥对。

```
nova keypair-add demo > demo.pem
chmod 0600 *.pem
```

3. 使用如下 nova 指令，测试一下密钥对是否创建成功。

```
nova list
nova credentials
```

工作原理

Nova 客户端需要使用 OpenStack 认证服务来进行认证和授权，这样它才能创建虚拟机实例。所以，这里手动为 OpenStack 认证服务创建了一个环境资源文件来保存所需的环境

变量。

运行时环境会将 `username`、`password` 和 `tenant` 这些环境变量传递给 OpenStack 认证服务进行验证，并在后台返回一个相应的令牌确认用户身份。这样一来，就可以在某个租户（或项目）中直接启动虚拟机实例了。

4.12 管理安全组

安全组是虚拟机实例的防火墙，在云环境中必须进行设置。该防火墙存在于运行 OpenStack 计算的主机上，而非虚拟机自身内部的 `iptables` 规则。它通过开放或禁止对指定服务端口的服务权限来保护主机，同时保护虚拟机实例不会被运行在同一主机上的其他实例攻击。如果当前环境是 Flat 网络模式而非 VLAN 或隧道（`tunnel`）模式，安全组几乎是唯一可以用来隔离不同租户之间实例的办法。

准备工作

首先确保登录到一台可以运行 Nova 客户端工具的客户机上。这可以用如下命令安装。

```
sudo apt-get update
sudo apt-get install python-novaclient
```

同时确认已经设置好下列认证信息。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=http://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

以下部分将介绍如何在 OpenStack 环境中创建和修改安全组。

创建安全组

使用 Nova 客户端运行以下命令，将 80 端口和 443 端口的 TCP 连接权限开放给名为 `webserver` 的安全组。

```
nova secgroup-create webserver "Web Server Access"
nova secgroup-add-rule webserver tcp 80 80 0.0.0.0/0
nova secgroup-add-rule webserver tcp 443 443 0.0.0.0/0
```

之所以新建一个安全组，而不是修改默认安全组的规则，是因为不希望向所有人开放 Web 服务器。如果直接修改默认安全组，每次创建实例时就会开放给所有人。通过将权限赋给一个特定的安全组，可以在创建实例时绑定该安全组从而开放实例的 80 端口。

当创建实例时，指定 `--security-groups` 选项，如下。

```
nova boot myInstance \
  --image 0e2f43a8-e614-48ff-92bd-be0c68da19f4
  --flavor 2 \
  --key_name demo \
  --security_groups default,webserver
```

从安全组中删除规则

运行 `nova secgroup-delete-rule` 可以从安全组中删除规则。例如，使用 Nova 客户端运行以下命令，将 HTTPS 规则从 `webserver` 安全组中删除。

```
nova secgroup-delete-rule webserver tcp 443 443 0.0.0.0/0
```

删除安全组

运行以下命令，删除 `webserver` 安全组。

```
nova secgroup-delete webserver
```

工作原理

创建一个安全组只需要两个步骤。首先使用 `nova secgroup-create` 命令创建一个安全组。创建完以后，使用 `nova secgroup-add-rule` 命令在安全组里面添加规则。通过该命令，可以指定虚拟机实例对哪些网络开放哪些端口。

使用 Nova 客户端定义安全组和规则

`nova secgroup-create` 命令的基本语法如下。

```
nova secgroup-create group_name "description"
```

`nova secgroup-add-rule` 命令的基本语法如下。

```
nova secgroup-add-rule group_name protocol port_from port_to source
```

要从安全组中删除一条规则，可使用 `nova secgroup-delete-rule` 命令，语法与 `nova secgroup-add-rule` 命令类似。删除安全组使用 `nova secgroup-delete` 命令，语法与 `nova secgroup-create` 命令类似。

4.13 创建和管理密钥对

SSH 密钥对包含公钥和私钥两部分。用 SSH 登录 Linux 主机时需要密钥对。公钥部分会在虚拟机实例启动时通过 cloud-init 服务注入实例。cloud-init 可以做很多工作，管理公钥的注入是其中的一种。只有公钥和私钥匹配，才能被允许访问虚拟机实例。

准备工作

首先确保登录到一台可以运行 Nova 客户端工具的 Ubuntu 客户机上。该工具可以用如下命令安装。

```
sudo apt-get update
sudo apt-get install python-novaclient
```

同时确认已经设置好下列认证信息。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=http://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

使用 nova keypair-add 创建一个密钥对。同时为这个密钥对取一个名字，这样在虚拟机实例启动时可以通过这个名字指定密钥对。该命令的输出结果是一个 SSH 私钥，访问虚拟机实例系统界面时将会用到它。

1. 首先创建一个密钥对，如下。

```
nova keypair-add demokey > demokey
```

2. 保护这个私钥文件，确保只有已登录的用户才能读取它。

```
chmod 0600 demokey
```

该命令生成一个密钥对，并将公钥保存在 OpenStack 环境核心的数据库中。私钥内容则写入客户机上的某个文件中。私钥文件要进行妥善保管，只允许虚拟机用户才能读取。

要在 Nova 客户端下使用这个新创建的密钥对时，可以这样执行 nova boot 命令。

```
nova boot myInstance \
  --image 0e2f43a8-e614-48ff-92bd-be0c68da19f4 \
```

```
--flavor 2 --key_name demokey
```

当需要用 SSH 登录这个运行中的实例时，在 SSH 命令中用 `-i` 选项指定私钥。

```
ssh ubuntu@172.16.1.1 -i demokey
```



与 Unix 环境下的大部分情况一样，这里的值和文件名都是区分大小写的。

使用 Nova 客户端列出和删除密钥对

要用 Nova 客户端列出和删除密钥对，执行下面几个小节中的命令。

列出密钥对

要使用 Nova 客户端列出项目中的密钥对，只要简单运行 `nova keypair-list` 即可，具体如下。

```
nova keypair-list
```

这会返回项目中的密钥对列表，如下所示。

Name	Fingerprint
demokey	77:ad:94:d6:8b:c6:d8:45:85:55:22:2b:ad:b3:22:e9

删除密钥对

要从项目中删除一个密钥对，只要在 `keypair-delete` 命令的选项中简单指定一下密钥对的名字即可。

❑ 删除 `myKey` 密钥对的命令如下。

```
nova keypair-delete demokey
```

❑ 使用以下命令验证执行结果。

```
nova keypair-list
```



删除密钥对是不可恢复的动作。删除一个运行中实例的密钥对将导致无法再访问该实例。

工作原理

密钥对在云计算环境中很重要，因为大部分 Linux 镜像不允许在命令行中输入用户名和密码。Cirros 镜像是一个例外，它默认的用户是 `cirros`，密码是 `cubswin :)`。

Cirros 是一个裁剪过的镜像，用于 OpenStack 的故障检测和测试。而像 Ubuntu 这样的镜像则只允许使用密钥对登录。

用 `nova keypair-add` 命令创建密钥对后，便可以通过 SSH 访问虚拟机实例了。该命令会将公钥保存在后端数据库中，该公钥会通过虚拟机实例启动或初始化时的脚本注入到实例的 `.ssh/authorized_keys` 文件中。以后只要在 `ssh` 命令行中使用 `-i` 选项指定相应的密钥文件就可以访问这个实例了。

当然，也可以将一个密钥对从项目中删除掉，从而停止这对密钥对实例的访问权限。这是用 `nova keypair-delete` 命令来完成的。可以运行 `keypair-list` 验证项目中可用的密钥对。

4.14 启动第一个云实例

至此，已有了一个 OpenStack 计算服务的运行环境以及一个可以使用的镜像，是时候启动第一个云实例了！本节将介绍如何使用通过 `nova image-list` 和 `nova flavor-list` 命令获取的信息来启动需要的实例。

准备工作

以下步骤要在一台 Ubuntu 机器上运行，用户需能访问（4.8 节所创建的）OpenStack 计算服务的证书。

确保登录到了网络节点，且该节点可访问因特网，能够按照在环境中运行 OVS 和 Neutron 所需的软件包。如果通过 Vagrant 创建了该节点，可执行如下命令。

```
vagrant ssh network
```

在启动第一个实例前，必须创建默认的安全设置，定义访问权限。该设置只需做一次（除非需要调整），可以在 Nova 客户端使用 `nova secgroup-add-rule` 命令完成。下面的命令集将允许用户从任意 IP 地址通过 SSH 访问（端口 22），而且可以 ping 实例来帮助故障排查。注意，如果命令行中没有特意说明，将使用默认的安全组规则。

具体步骤如下。

1. 在安装有 Nova 客户端的环境中，执行以下动作配置合适的环境变量。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
```



```
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```



将这几行命令加入到 `home` 目录的 `novarc` 文件。这样每次只需执行 `source novarc` 命令即可简单地导入这些认证信息。

2. 使用 Nova 客户端，通过以下命令增加一些合适的规则。

```
nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
```



如果还没有可用的镜像，可参照第 2 章 2.4 节中的步骤创建一个。

操作步骤

环境就绪后，执行以下指令来启动第一个虚拟机实例。

1. 执行以下命令列出所有可用的镜像：

```
nova image-list
```

该命令输出的内容如图 4-7 所示。

ID	Name	Status	Server
43f974b7-1f74-4305-8354-0ac0c3efe68d	cirros-image	ACTIVE	
5bfb4a6d-da77-4502-ba52-2e8e40597e96	trusty-image	ACTIVE	

图 4-7

2. 然后，执行如下命令获取可用的镜像类型（可以视作大小）：

```
nova flavor-list
```

该命令的输出如图 4-8 所示。

ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor	Is_Public
1	m1.tiny	512	0	0		1	1.0	True
2	m1.small	2048	20	0		1	1.0	True
3	m1.medium	4096	40	0		2	1.0	True
4	m1.large	8192	80	0		4	1.0	True
5	m1.xlarge	16384	160	0		8	1.0	True

图 4-8

可以通过 ID 或名称指定需要使用的类型。

3. 由于实验环境配置为使用两个网络，需要选择将实例添加到哪一个。首先，使用如

下命令列出可用的网络：

```
neutron net-list
```

该命令的输出如图 4-9 所示。

id	name	subnets
0375e772-b021-425c-bc17-5a3263247fb8	cookbook_network_1	37ca3149-ee6b-4288-b074-03a4e1635b7c 10.200.0.0/24
706c9118-68d9-4751-932c-3dc94ec6f4ed	ext_net	889c1ef7-09af-48d2-85bf-6971446e2eb7 192.168.100.0/24

图 4-9

4. 启动实例时，需要指定镜像、类型、网络和密钥名称等之前在命令行获取到的信息。如要通过 Nova 客户端工具启动实例，可以在执行如下命令时提供 `trusty-image` 和 `cookbook_network_1` 的镜像的 UUID。

```
nova boot myInstance \
  --image 5bfb4a6d-da77-4502-ba52-2e8e40597e96 \
  --flavor 2 \
  --nic net-id=0375e772-b021-425c-bc17-5a3263247fb8 \
  --key_name demokey
```

之后再启动实例时，会看到类似图 4-10 的输出。

Property	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-SRV-ATTR:host	-
OS-EXT-SRV-ATTR:hypervisor_hostname	-
OS-EXT-SRV-ATTR:instance_name	instance-00000002
OS-EXT-STS:power_state	0
OS-EXT-STS:task_state	scheduling
OS-EXT-STS:vm_state	building
OS-SRV-USG:launched_at	-
OS-SRV-USG:terminated_at	-
accessIPv4	
accessIPv6	
adminPass	MgkrXAVjbpM8
config_drive	
created	2015-01-29T06:12:21Z
flavor	m1.small (2)
hostId	
id	5971ab77-9d91-40d8-9961-d86da0945f26
image	trusty-image (5bfb4a6d-da77-4502-ba52-2e8e40597e96)
key_name	demokey
metadata	{}
name	myInstance
os-extended-volumes:volumes_attached	[]
progress	0
security_groups	default
status	BUILD
tenant_id	e99b80a0de78451f91c97beef5b2c2d5
updated	2015-01-29T06:12:21Z
user_id	183fd6a93d7e4a65aa513bdb4fa9850e

图 4-10

5. 启动实例需要一点时间。可通过如下命令检查实例的状态。

```
nova list
nova show 5971ab77-9d91-40d8-9961-d86da0945f26
```

6. 上一个命令的输出类似此前命令的输出。但是，这一次成功创建了一个实例，并且运行并分配了一个 IP 地址，如图 4-11 所示。

Property	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-SRV-ATTR:host	compute-02
OS-EXT-SRV-ATTR:hypervisor_hostname	compute-02.cook.book
OS-EXT-SRV-ATTR:instance_name	instance-00000002
OS-EXT-STS:power_state	1
OS-EXT-STS:task_state	-
OS-EXT-STS:vm_state	active
OS-SRV-USG:launched_at	2015-01-29T06:12:47.000000
OS-SRV-USG:terminated_at	-
accessIPv4	
accessIPv6	
config_drive	
cookbook_network_1 network	10.200.0.5
created	2015-01-29T06:12:21Z
flavor	m1.small (2)
hostId	c8b5ccd18b4c6930917d8e6a4b26e4ebf3722f9e6d2515a72991e1ef
id	5971ab77-9d91-40d8-9961-d86da0945f26
image	trusty-image (5bfb4a6d-da77-4502-ba52-2e8e40597e96)
key_name	demokey
metadata	{}
name	myInstance
os-extended-volumes:volumes_attached	[]
progress	0
security_groups	default
status	ACTIVE
tenant_id	e99b80a0de78451f91c97beef5b2c2d5
updated	2015-01-29T06:12:47Z
user_id	183fd6a93d7e4a65aa513bdb4fa9850e

图 4-11

7. 稍等片刻，就可以访问刚刚创建的实例。如果使用的是 Vagrant 环境，从网络节点上即可通过网络空间和 SSH 私钥访问实例。首先，使用如下命令列出网络空间。

```
ip netns
```

该命令的输出如下。

```
qdhcp-0375e772-b021-425c-bc17-5a3263247fb8
```

现在，通过如下命令连接到实例。

```
sudo ip netns exec \
    qdhcp-0375e772-b021-425c-bc17-5a3263247fb8 \
    ssh -i demokey ubuntu@10.200.0.5
```



Ubuntu 云镜像中内建的默认用户为 ubuntu。

祝贺！已经成功启动并连接到第一个 OpenStack 云实例。

工作原理

创建默认的安全设置之后，先记录下镜像的标识符——一个 UUID 值，然后通过 Nova

客户端工具启动实例。命令行中同时指定了所要使用的密钥对。接着用生成的密钥对中的私钥来访问实例。

云实例如何知晓使用什么密钥？作为该镜像启动脚本的一部分，它会回调 meta-server，它是 nova-api 和 nova-api-metadata 服务的一个功能。这个 meta-server 在实例和真实世界的云环境中提供了一个中间桥梁，并且能在云初始化启动过程被调用。在这里，它会下载一个脚本将私钥注入到 Ubuntu 用户的 .ssh/authorized_keys 文件中。后面将会介绍如何在修改启动过程中调用脚本。

当一个云实例启动后，它会生成许多关于实例的大量的指标和细节，这些信息可以通过 nova list 和 nova show 命令获取。nova list 命令显示一个简洁版本的信息，仅列出实例的 ID、名字、状态和 IP 地址。

本例在 nova boot 命令中选择了 ID 为 2 的实例类型。可以通过以下命令列出支持的实例类型：

```
nova flavor-list
```

这些类型如表 4-1 所示。

表 4-1

实例类型	内存	VCPUS	存储	版本
m1.tiny	512 MB	1	1 GB	32 位和 64 位
m1.small	2048 MB	1	20 GB	32 位和 64 位
m1.medium	4096 MB	2	40 GB	仅 64 位
m1.large	8192 MB	4	80 GB	仅 64 位
m1.xlarge	16384 MB	8	160 GB	仅 64 位

4.15 修复出错的实例部署

部署实例时，有时会出错导致部署失败。如果在部署新实例时出现该问题，通过最简单的方法是删除失败的实例然后重新部署。但是，如果需要修复出错的实例，可以使用 nova rescue 命令。本节解释如何使用 nova rescue 命令修复出错的实例。

准备工作

如果使用的是 Vagrant 环境, 本节的操作步骤将在网络节点下进行, 用户需能访问 (4.8 节所创建的) OpenStack 计算服务的证书。

确保已经登录到网络节点, 且该节点可访问因特网, 能够按照在环境中运行 OVS 和 Neutron 所需的软件包。如果通过 Vagrant 创建了该节点, 可执行如下命令。

```
vagrant ssh network
```

操作步骤

在网络节点上, 列出所有正在运行的实例, 找到需要修复的实例, 并执行如下步骤。

1. 首先在客户机上使用如下命令, 找到要修复的实例。

```
nova list
```

2. 指定实例的名称或 UUID, 将实例设置为 rescue 模式:

```
nova rescue myInstance
nova rescue 6f41bb91-0f4f-41e5-90c3-7ee1f9c39e5a
```

运行上述命令后, 将会获得一个临时 root 密码。

```
+-----+-----+
| Property | Value |
+-----+-----+
| adminPass | VMHm2BEyCnKa |
+-----+-----+
```

但是, 如果使用 nova 密钥创建了该实例, 仍需使用私钥而不是密码, 才能登陆到实例。这时, 实例应该处于 RESCUE 状态。

3. 登录到节点之前, 首先获取网络空间列表。

```
ip netns
```

上述命令会得到如下输出。

```
qdhcp-0375e772-b021-425c-bc17-5a3263247fb8
```

接下来, 可使用如下命令连接到实例。

```
sudo ip netns exec \
    qdhcp-0375e772-b021-425c-bc17-5a3263247fb8 \
    ssh -i demokey ubuntu@10.200.0.5
```

之后可以开始修复实例了。

4. 修复好之后, 从正常的启动磁盘重启服务器。

```
nova unrescue myInstance
```

工作原理

我们通过 nova list 命令找到了需要修复的服务器。然后, 使用服务器名称或 ID 将

服务器设置为 rescue 模式，这是通过 `nova rescue` 命令实现的。这会重启服务器为 rescue 模式，即从最初的镜像启动服务器，然后将当前的启动磁盘更改为次级磁盘。在修复服务器之后，需要将其调整回 ACTIVE 状态，并通过 `nova unrescue` 命令重启。

4.16 终止实例

云环境的动态特性意味着云实例能够按需启动和终止。终止一个实例非常简单，重要的是要来理解云实例的一些基本概念。

某些云实例，比如本书使用的实例不是持久化的。这意味着数据和该实例所做的工作只存在于本次运行时。一个云实例可以被重启，一旦终止，所有的数据都会丢失。



为保证不丢失数据，OpenStack 块存储服务 Cinder 提供数据持久存储功能，允许附加一个卷到运行的实例上，它不会在实例被终止时销毁。一个卷就像一个 USB 驱动器连接到实例。更多信息查看第 8 章。

操作步骤

从 Ubuntu 机器上列出正在运行的实例，找到想终止的实例。

1. 首先在客户端执行以下命令找到希望终止的实例。

```
nova list
```

2. 通过指定实例的名字或者 UUID 来删除实例。

```
nova delete myInstance
```

```
nova delete 6f41bb91-0f4f-41e5-90c3-7ee1f9c39e5a
```

可以再次运行 `nova list` 来验证一下实例是否已经被删除了。



在实例名字和 UUID 之间，推荐使用 UUID，因为可以避免名字混淆。另外，如果存在多个重名的实例，可能会出现无法预料的结果。使用 UUID 不会出现这种问题。

工作原理

首先，在 `novalist` 命令的结果中通过名字或 UUID 找到希望终止的实例；然后，使用 `nova delete` 删除它。一旦终止，该实例将不再存在，意味着将被销毁。所以，如果该

实例中存有任何数据，也将会随着实例被删除。

4.17 使用在线迁移

OpenStack Nova 支持在计算主机之间在线迁移基于虚拟机的实例。这对于维护和集群均衡很有帮助。在线迁移之前，必须首先将节点添加到 Nova 集群。

准备工作

假设有多个运行 Nova 计算服务的主机，并按照 4.5 节进行了配置。如果不是这样，在继续之前需要先配置第二台计算主机。目标主机还需要能够远程访问，且有足够的资源运行将要迁移的实例。

检查网络连接

为了成功完成在线迁移，两台主机必须能够通过主机名称（hostname）通信。可以登录到每台主机，并执行 ping 命令进行验证。

```
$ ping compute-02
PING compute-02.book (192.168.100.203) 56(84) bytes of data.
64 bytes from compute-02.book (192.168.100.203): icmp_seq=1 ttl=64
time=2.14 ms
64 bytes from compute-02.book (192.168.100.203): icmp_seq=2 ttl=64
time=0.599 ms
$ ping compute-01
PING compute-01.book (192.168.100.202) 56(84) bytes of data.
64 bytes from compute-01.book (192.168.100.202): icmp_seq=1 ttl=64
time=1.29 ms
64 bytes from compute-01.book (192.168.100.202): icmp_seq=2 ttl=64
time=0.389 ms
```

确保资源充足

在线迁移还有赖于远程主机有足够的资源完成工作负载。可以通过 nova 命令行工具登陆主机，并使用如下 nova 命令进行确认。

```
$ nova host-describe compute-02
```

HOST	PROJECT	cpu	memory_mb	disk_gb
compute-02	(total)	2	3107	37
compute-02	(used_now)	0	512	0
compute-02	(used_max)	0	0	0

第一行列出了主机上可用的资源总量。在本例中，主机上有两个 vCPU，3 GB 内存和

37 GB 磁盘。总量行中的数减去 `used_now` 行的数，就可以得到主机上可用的资源。本例中，目前使用的只有 512 MB 的内存，即还有 2.5 GB 可用于迁移。

操作步骤

在继续之前，请通过 Nova 命令行工具登陆到一台主机。执行如下命令，可在节点之间迁移基于 VM 的实例。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/

nova live-migration --block-migrate <UUID> compute-02
```

工作原理

在线迁移是 OpenStack 的一个核心特性，可让 OpenStack 维护者和管理员在管理云基础设施时，不对云消费者造成影响。另外，管理员可使用 Ceilometer 的监控数据，做出合理的在线迁移选择，保证 OpenStack 云环境的工作负载均衡。

在 OpenStack 中，在线迁移是由 libvirt 驱动处理的。具体来说，在发出 `nova live-migration` 命令时，OpenStack 计算服务会在一个计算主机上创建一个 libvirtd 连接到远程主机的相同进程。连接建立后，根据所设定的参数不同，会同步实例的内存状态，转移实例的控制权。在本例中，我们指定了额外的 `--block-migrate` 参数，指的是在没有共享存储的情况下如何处理实例的磁盘文件。

4.18 使用 nova-scheduler

在启动 OpenStack 实例时，`nova-scheduler` 的工作是确定在哪个计算主机（管理程序）上创建该实例。调度器可以进行设置，实现一些基本的决策，如判断是否存在内存，或者是否存在足够的 CPU 核心等。还可以做更复杂的配置，基于环境因素和元数据做出决策，这样可以让实例归集到某些主机上，或者分布到不同的主机上，确保在计算主机故障时环境仍然稳定运行。

准备工作

确保已经登录到了 OpenStack 控制节点。如果使用 Vagrant 创建了该节点，可执行如下命令。

```
vagrant ssh controller
```

操作步骤

修改/etc/nova/nova.conf 文件，开启上面提到的所有调度器过滤器。

1. 将下面这些行内容，添加到控制节点的/etc/nova/nova.conf 文件的[Default]部分。

```
scheduler_driver=nova.scheduler.multi.MultiScheduler
scheduler_driver_task_period = 60
scheduler_driver = nova.scheduler.filter_scheduler.FilterScheduler
scheduler_available_filters = nova.scheduler.filters.all_filters
scheduler_default_filters = RetryFilter, AvailabilityZoneFilter,
    RamFilter, ComputeFilter, ComputeCapabilitiesFilter,
    ImagePropertiesFilter, ServerGroupAntiAffinityFilter
    ServerGroupAffinityFilter
```

2. 重启 nova-scheduler 服务，使得改动生效。

```
sudo service nova-scheduler restart
```

工作原理

修改 nova.conf 文件可以启用 nova-scheduler 的其他特性。在本例中，OpenStack Juno 版本默认的调度器在满足以下条件时，会考虑调度一个实例到计算主机。

- ☐ RetryFilter: 重试每个计算主机（在第一次时，默认此前没有请求过主机）。
- ☐ AvailabilityZoneFilter: 要求主机处于请求的可用区（默认是 nova）。
- ☐ RamFilter: 要求计算主机有足够的 RAM 内存。
- ☐ ComputeFilter: 要求计算主机可用，能够响应请求。
- ☐ ComputeCapabilitiesFilter: 要求计算主机满足请求的实例所需要的其他条件。
- ☐ ImagePropertiesFilter: 要求请求的镜像（和相关的特性）可以在指定的主机上运行。
- ☐ ServerGroupAntiAffinityFilter: 如果收到请求，判断实例运行的主机是否要不同于同服务器中的其他实例。
- ☐ ServerGroupAffinityFilter: 如果收到请求，判断实例是否应该运行在属于同一实例的主机上。

更多参考

不同场景下，有不同类型的调度器可以选择。更多信息，请访问 http://docs.openstack.org/juno/config-reference/content/section_compute-scheduler.html。

4.19 创建实例类型

实例类型与大小有关。一般涉及 CPU 核数（虚拟 CPU）、内存大小，以及实例可用的本地或临时磁盘资源。标准的实例类型通常包括 m1.tiny、m1.small、m1.large 和 m1.xlarge 等。用户在命令行或 Horizon 界面中指定选择哪种实例类型。

准备工作

确保登录的 Ubuntu 主机可以访问 192.168.100.0/24 网络上的 OpenStack 环境。该主机将用于运行 OpenStack 环境相关的客户端工具。如果使用配套的 Vagrant 环境，可以使用控制节点。该节点上已经安装了 python-novaclient 软件包，提供了 swift 命令行客户端。

如果使用 Vagrant 创建了该节点，可以执行如下命令。

```
vagrant ssh controller
```

确保已经设置好了以下证书信息（如果没有使用 Vagrant 环境，请根据具体情况调整证书和密钥文件的路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/akey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

执行如下步骤，新建一个实例类型，具体为 2 核虚拟 CPU，16GB 内存和 30GB 磁盘空间。

1. 执行如下命令：

```
nova flavor-create m1.javaserver
49 16384 30 2
--is-public=true
```

上述命令的输出如图 4-12 所示。

ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor	Is_Public
49	m1.javaserver	16384	30	0		2	1.0	True

图 4-12

2. 现在通过如下命令，列出现有的实例类型。

```
nova flavor-list
```

上述命令的输出如图 4-13 所示。

ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor	Is_Public
1	m1.tiny	512	0	0		1	1.0	True
2	m1.small	2048	20	0		1	1.0	True
3	m1.medium	4096	40	0		2	1.0	True
4	m1.large	8192	80	0		4	1.0	True
49	m1.javaserver	16384	30	0		2	1.0	True
5	m1.xlarge	16384	160	0		8	1.0	True

图 4-13

工作原理

新建实例类型的语法如下。

```
nova flavor-create $FLAVOR_NAME
    $FLAVOR_ID $RAM $DISK $CPU
    --is-public={true|false}
    --ephemeral $EPHEMERAL_SIZE_GB
    --swap $SWAP_SIZE_GB
    --rxtx-factor $FACTOR
```

nova flavor-create 命令不会自动更新实例类型的 ID；因此必须手动指定 ID，并且确保是唯一的。然后指定内存、磁盘和 CPU。下面是可选的其他选项。

- ☐ --is-public=true|false：指定实例类型是否只对当前租户可用，或是对所有租户客户。只有管理员才能设置该选项。默认的值是 true。
- ☐ --ephemeral \$EPHEMERAL_SIZE_GB：允许指定一个额外的临时磁盘。
- ☐ --swap \$SWAP_SIZE_GB：指定与该实例相关联的附加交换分区。
- ☐ --rxtx-factor：指定该实例相对于其他类型的带宽比例。默认值是 1，值为 0.5 表示只有一半的带宽可用。

4.20 定义主机分组

主机分组（Host Aggregate）允许将硬件进行逻辑分组，并在部署时创建分区。这通常用于将相同规格的硬件归类在一起，如具备某类硬件（如 SSD 盘）的计算主机。然后可以定义与硬件分组相关的额外信息，也称为元数据，用户在启动实例时可以获取这些数据。例如，用户可以启动一个实例，指定希望在具备 SSD 硬盘的计算主机上运行。通过提供这个额外信息，收到该元数据的计算主机将会请求该实例在相应的硬件上启动。

计算主机还可以属于多个主机分组，在定义分区时可取得更大的灵活性，支持以多种

方式组织计算主机。图 4-14 就是一个使用主机分组的示例，定义了多个计算主机分组。只有管理员才能创建主机分组。

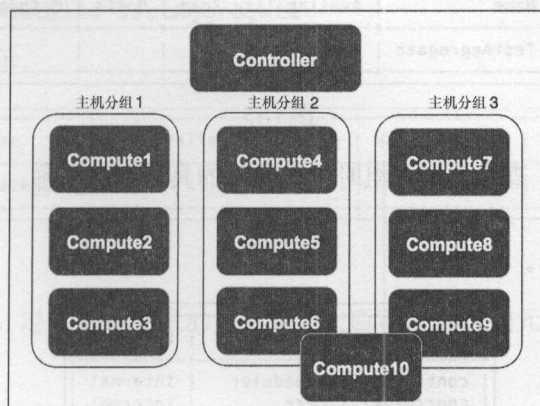


图 4-14

准备工作

确保登录的 Ubuntu 主机可以访问 192.168.100.0/24 网络上的 OpenStack 环境。该主机将用于运行 OpenStack 环境相关的客户端工具。如果使用配套的 Vagrant 环境，可以使用控制节点。该节点上已经安装了 python-novaclient 软件包，提供了 swift 命令行客户端。

如果使用 Vagrant 创建了该节点，可以执行如下命令：

```
vagrant ssh controller
```

确保已经设置好了以下证书信息（如果没有使用 Vagrant 环境，请根据具体情况调整证书和密钥文件的路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

执行如下步骤，创建一个名称为 TestAggregate 的主机分组，其中包括一台叫做 compute-02 的计算主机。

1. 首先使用如下命令，创建主机分组。

```
nova aggregate-create TestAggregate
```


输出如图 4-15 所示。

Id	Name	Availability Zone	Hosts	Metadata
1	TestAggregate	-		

图 4-15

2. 执行如下命令，查找主机分组时应该使用的具体主机名称。

```
nova host-list
```

输出如图 4-16 所示。

host_name	service	zone
controller	scheduler	internal
controller	cert	internal
controller	consoleauth	internal
controller	conductor	internal
compute-01	consoleauth	internal
compute-01	compute	nova
compute-02	consoleauth	internal
compute-02	compute	nova

图 4-16

3. 创建完主机分组，并记录下主机名称之后，只需要这样做就可以将主机添加至分组中。

```
nova aggregate-add-host TestAggregate compute-02
```

输出如图 4-17 所示。

Host compute-02 has been successfully added for aggregate 1				
Id	Name	Availability Zone	Hosts	Metadata
1	TestAggregate	-	'compute-02'	

图 4-17

4. 然后，即可使用如下命令列出环境中可用的主机分组。

```
nova aggregate-list
```

输出如图 4-18 所示。

Id	Name	Availability Zone
1	TestAggregate	-

图 4-18

5. 还可以执行如下命令，获取有关分组的更多细节，如分组中都有什么主机，主机相关的元数据，分组是否与可用区关联等。

```
nova aggregate-details TestAggregate
```

输出如图 4-19 所示。

Id	Name	Availability Zone	Hosts	Metadata
1	TestAggregate	-	'compute-02'	

图 4-19

6. 使用如下命令，定义主机分组的元数据，之后可使用这些元数据指定实例在该分组的主机上启动。

```
nova aggregate-set-metadata TestAggregate highspec=true
```

这会设置 highspec=true，稍后会用到。命令的输出如图 4-20 所示。

Metadata has been successfully updated for aggregate 1.				
Id	Name	Availability Zone	Hosts	Metadata
1	TestAggregate	-	'compute-02'	'highspec=true'

图 4-20

7. 然后可以通过用来启动实例的实例类型暴露该元数据。作为演示，我们创建一个名为 m1.highspec 的实例类型，将类型的元数据设置为与主机分组相同。使用如下命令，即可创建一个名为 m1.highspec 的新类型：

```
nova flavor-create
m1.highspec
50 2048 20 2
--is-public=true
```

8. 接着将该类型的元数据，与 TestAggregate 分组中的元数据相匹配。

```
nova flavor-key m1.highspec set highspec=true
```

9. 然后，可以通过如下命令，查看实例类型的细节。

```
nova flavor-show m1.highspec
```

命令的输出如图 4-21 所示。

我们可以在启动实例时指定为该类型。这会自动调度 TestAggregate 分组中的一台主机去启动实例，本例中会调度 compute-02 主机。

Property	Value
OS-FLV-DISABLED:disabled	False
OS-FLV-EXT-DATA:ephemeral	0
disk	20
extra_specs	{"highspec": "true"}
id	50
name	m1.highspec
os-flavor-access:is_public	True
ram	2048
rxtx_factor	1.0
swap	
vcpus	2

图 4-21

工作原理

主机分组可以让管理员以对终端用户透明的方式定义计算资源，同时根据目的进行逻辑地分组。当主机分组的元数据与实例类型的元数据匹配，并且使用该类型启动实例时，就会调度主机分组中的计算主机去创建、运行该实例。

4.21 在特定可用区启动实例

可用区是对计算资源的逻辑隔离，代表用户启动实例时可以选择的管理程序组。如果没有创建过可用区，将使用名为 **nova** 的默认可用区。实例以默认设置启动时，调度器会决定由可用区中哪台主机运行。如果有多个可用区，OpenStack 云环境的用户可以指定使用哪个可用区。这有助于创建更加灵活的应用。在两个不同的可用区启动实例，可以避免某个可用区故障带来的问题。

准备工作

确保登录的 Ubuntu 主机可以访问 192.168.100.0/24 网络上的 OpenStack 环境。该主机将用于运行 OpenStack 环境相关的客户端工具。如果使用配套的 Vagrant 环境，可以使用控制节点。该节点上已经安装了 python-novaclient 软件包，提供了 nova 命令行客户端。

如果使用 Vagrant 创建了该节点，可以执行如下命令。

```
vagrant ssh controller
```

确保已经设置好了以下证书信息（如果没有使用 Vagrant 环境，请根据具体情况调整证书和密钥文件的路径）。

```
export OS_TENANT_NAME=cookbook
```



```
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

1. 使用如下语法，在特定可用区启动一个实例。

```
nova boot
  --flavor $FLAVOR
  --image $IMAGE
  --availability-zone $AZ
  $INSTANCE_NAME
```

2. \$AZ 的名字来自 nova hypervisor-list，创建时使用命令显示的全名。

如果要在 compute-02 可用区启动一个名为 myInstance 的实例，执行如下命令：

```
nova hypervisor-list
```

命令输出如下。

```
+-----+
| ID | Hypervisor hostname |
+-----+
| 1 | compute-01.cook.book |
| 2 | compute-02.cook.book |
+-----+
```

3. 然后，使用如下命令，在 compute-02 中启动该实例。

```
nova boot
  --flavor 1
  --image trusty-image
  --availability-zone nova:compute-02.cook.book
  MyInstance
```



注意，OpenStack 只有在剩余足够的 CPU 核数和 RAM 时，并且满足限额要求时，才能够成功在特定可用区启动实例。如果没有足够的资源，系统会报错，提示没有可用的主机。

工作原理

在 nova 启动命令行中，使用如下标志即可在特定可用区启动实例。

```
--availability-zone $NAME_OF_ZONE
```

可用区的名称可使用下面的命令查询。

nova availability-zone-list

该命令会列出环境中所有可用的可用区。

4.22 在特定计算主机启动实例

大部分情况下，实例启动时，调度器会决定由哪台主机运行。但是，在进行错误排查或者编排服务管理资源分配时，最好是能够直接指定实例至某个主机运行。

准备工作

确保登录的 Ubuntu 主机可以访问 192.168.100.0/24 网络上的 OpenStack 环境。该主机将用于运行 OpenStack 环境相关的客户端工具。如果使用配套的 Vagrant 环境，可以使用控制节点。该节点上已经安装了 python-novaclient 软件包，提供了 nova 命令行客户端。

如果使用 Vagrant 创建了该节点，可以执行如下命令：

```
vagrant ssh controller
```

确保已经设置好了以下证书信息（如果没有使用 Vagrant 环境，请根据具体情况调整证书和密钥文件的路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

1. 使用如下语法，在特定主机上启动实例。

```
nova boot
--flavor $FLAVOR
--image $IMAGE
--availability-zone nova:$HYPERVISOR_NAME
$INSTANCE_NAME
```

\$INSTANCE_NAME 的名字来自 nova hypervisor-list，创建时使用命令显示的全名。

2. 如果要在 compute-02 可用区启动一个名为 myInstance 的实例，执行如下命令：

```
nova hypervisor-list
```

命令输出如下。

```
+-----+
| ID | Hypervisor hostname |
+-----+
| 1 | compute-01.cook.book |
| 2 | compute-02.cook.book |
+-----+
```

3. 然后, 使用如下命令, 在 compute-02 中启动该实例。

```
nova boot
--flavor 1
--image trusty-image
--availability-zone nova:compute-02.cook.book
myInstance
```



注意, OpenStack 只有在剩余足够的 CPU 核数和 RAM 时, 并且满足限额要求时, 才能够成功在特定主机上启动实例。如果没有足够的资源, 系统会报错, 提示没有可用的主机。

工作原理

在 nova 启动命令行中, 使用如下标志即可在特定主机启动实例。

```
--availability-zone nova: $HYPERVISOR_NAME
```

可用区的名称可使用下面的命令查询。

```
nova hypervisor-list
```

该命令会列出环境中所有可用的可用区。

4.23 从集群移除 Nova 节点

有时需要从集群中移除一个计算节点, 进行错误排查或维护。这时应该谨慎, 因为可能会对正在运行的虚拟机造成负面影响。在开始之前, 确保计算集群中有足够的资源, 在移除节点前将正在运行的虚拟机迁移到其他计算主机。

准备工作

确保登录的 Ubuntu 主机可以访问 192.168.100.0/24 网络上的 OpenStack 环境。该主机将用于运行 OpenStack 环境相关的客户端工具。如果使用配套的 Vagrant 环境, 可以使用控制节点。该节点上已经安装了 python-novaclient 软件包, 提供了 nova 命令行客户端。

如果使用 Vagrant 创建了该节点, 可以执行如下命令。

vagrant ssh controller

确保已经设置好了以下证书信息（如果没有使用 Vagrant 环境，请根据具体情况调整证书和密钥文件的路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/akey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

将使用 nova 命令行客户端禁用服务，并迁移虚拟机。

1. 使用 nova service-list 命令确定停止哪项 nova 服务。

```
nova service-list
```

命令输出如图 4-22 所示。

Id	Binary	Host	Zone	Status	State	Updated_at	Disabled Reason
1	nova-scheduler	controller	internal	enabled	up	2015-02-09T05:16:34.000000	-
2	nova-consoleauth	controller	internal	enabled	up	2015-02-09T05:16:41.000000	-
3	nova-cert	controller	internal	enabled	up	2015-02-09T05:16:40.000000	-
4	nova-conductor	controller	internal	enabled	up	2015-02-09T05:16:43.000000	-
5	nova-consoleauth	compute-01	internal	enabled	up	2015-02-09T05:16:37.000000	-
6	nova-compute	compute-01	nova	enabled	up	2015-02-09T05:16:37.000000	-
7	nova-consoleauth	compute-02	internal	enabled	up	2015-02-09T05:16:40.000000	-
8	nova-compute	compute-02	nova	enabled	up	2015-02-09T05:16:43.000000	-

图 4-22

2. 还需要 compute-01 主机上运行的所有虚拟机的名单。使用如下命令，搜索主机名称获取名单。

```
nova list --host compute-01
```

在本例中，如图 4-23 所示只有一个虚拟机 test1。

ID	Name	Status	Task State	Power State	Networks
5ee8002a-c658-4a33-8ddd-11af50b2e18c	test1	ACTIVE	-	Running	cookbook_network_1=10.200.0.6

图 4-23

3. 然后，需要使用如下命令停用 compute-01。


上述命令的输出如下所示。

Host	Binary	Status
compute-01	nova-compute	disabled

4. 现在, 使用 `nova service-list` 命令检查正在运行的服务时, 会发现 `compute-01` 已经被停用了, 如图 4-24 所示。

Id	Binary	Host	Zone	Status	State	Updated_at	Disabled Reason
1	nova-scheduler	controller	internal	enabled	up	2015-02-09T05:16:34.000000	-
2	nova-consoleauth	controller	internal	enabled	up	2015-02-09T05:16:41.000000	-
3	nova-cert	controller	internal	enabled	up	2015-02-09T05:16:40.000000	-
4	nova-conductor	controller	internal	enabled	up	2015-02-09T05:16:43.000000	-
5	nova-consoleauth	compute-01	internal	enabled	up	2015-02-09T05:16:37.000000	-
6	nova-compute	compute-01	nova	disabled	up	2015-02-09T05:16:40.000000	-
7	nova-consoleauth	compute-02	internal	enabled	up	2015-02-09T05:16:40.000000	-
8	nova-compute	compute-02	nova	enabled	up	2015-02-09T05:16:43.000000	-

图 4-24

 注意, 即使计算主机被停用, 并不意味着主机没有在运行。仍然可能会有虚拟机在这些主机上运行, 如果这样的话, 应该进行迁移。

5. 然后需要迁移 `compute-01` 上运行的所有虚拟机。在每个虚拟机上使用 `nova migrate` 命令。

```
nova migrate test1
```

该命令没有输出。在本例中, 只有一台需要迁移的虚拟机 `test1`。如果存在其他需要迁移的虚拟机, 请做相应迁移。

6. 使用 `nova show` 命令检查迁移虚拟机的状态。

```
nova show test1
```

7. 虚拟机在迁移时, 应该会看到如图 4-25 所示输出。

在进行下一步之前, 需要等待虚拟机进入 `VERIFY_RESIZE` 状态。

8. 验证虚拟机成功完成了迁移, 通过 `nova resize-confirm` 命令确认重新变更大小。

```
nova resize-confirm test1
```

现在已经成功停用 `compute-01` 主机, 主机上也没有任何虚拟机。接着可以关闭电源, 或在节点上执行维护任务。

Property	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-SRV-ATTR:host	compute-01
OS-EXT-SRV-ATTR:hypervisor_hostname	compute-01.cook.book
OS-EXT-SRV-ATTR:instance_name	instance-00000004
OS-EXT-STS:power_state	1
OS-EXT-STS:task_state	resize_migrating
OS-EXT-STS:vm_state	active
OS-SRV-USG:launched_at	2015-02-02T08:06:45.000000
OS-SRV-USG:terminated_at	-
accessIPv4	
accessIPv6	
config_drive	
cookbook_network_1 network	10.200.0.6
created	2015-02-02T08:06:37Z
flavor	m1.tiny (1)
hostId	80e19db4c2ee0ac10520eedbd960227393fa6467b300bf4db039d738
id	5ee8002a-c658-4a33-8ddd-11af50b2e18c
image	trusty-image (5bfb4a6d-da77-4502-ba52-2e8e40597e96)
key_name	demokey
metadata	{}
name	test1
os-extended-volumes:volumes_attached	[]
progress	0
security_groups	default
status	RESIZE
tenant_id	e99b80a0de78451f91c97beef5b2c2d5
updated	2015-02-09T04:36:23Z
user_id	183f66a93d7e4a65aa513bdb4fa9850e

图 4-25

工作原理

如果要从集群移除一个计算节点，首先需要确保集群的其他节点上有足够的资源，可以满足该节点上所有虚拟机的需求。然后需要使用 `nova service-disable` 命令停用该节点。

执行 `nova service-disable $SERVICE nova-compute` 命令，停用计算节点。之后，需要将该节点上的所有虚拟机迁移到其他计算节点。`Nova migrate` 命令可以将虚拟机迁移到资源充足的其他节点。

第 5 章

Swift——OpenStack 对象存储

本章将讲述以下内容：

- ☐ 在 Keystone 中配置 Swift 服务和用户
- ☐ 安装 OpenStack 对象存储——代理服务器
- ☐ 配置 OpenStack 对象存储——代理服务器
- ☐ 安装 OpenStack 对象存储服务——存储节点
- ☐ 配置 Swift 使用物理存储
- ☐ 配置对象存储备份
- ☐ 配置 OpenStack 对象存储——存储服务
- ☐ 制作对象存储环
- ☐ 停止和启动 OpenStack 对象存储
- ☐ 设置 SSL 访问

5.1 简介

OpenStack 对象存储 (Object Storage)，即 **Swift**，是基于通用硬件的高可扩展与高冗余的存储服务。此服务被 Rackspace 实现为云文件，本身也类似于 Amazon 的 S3 存储服务，而且在 OpenStack 下的管理方式也与 Amazon 的 S3 存储服务相似。基于 OpenStack 对象存储可以存储海量的对象文件，并且能够随时按需扩展以便容纳更多的存储需求。OpenStack 对象存储的高冗余特性对于归档数据（如日志）来说是非常理想的，同样也非常适合用来为 OpenStack 计算服务中的虚拟机实例模板镜像提供存储服务。

本章将配置一个多节点的测试环境，包括 1 个 Swift 代理服务器，5 个 Swift 存储节点，并且其对象存储将写入磁盘/dev/sbd1。验证和授权由 Keystone 负责。Swift 中存储的数据会备份 3 次，这意味着大小 1GB 的文件实例会使用集群中 3 GB 的空间。Swift 会将该数据分布于 5 个节点中（其中任意 3 个将会存储数据），保证任何一个节点出现故障时可以快速地恢复。这可以保证不管哪个节点故障，都不会出现问题。这也是 Swift 安装时最低要求的推荐架构。

下图 5-1 是典型的参考架构，其中代理服务器位于一个负载均衡之后。

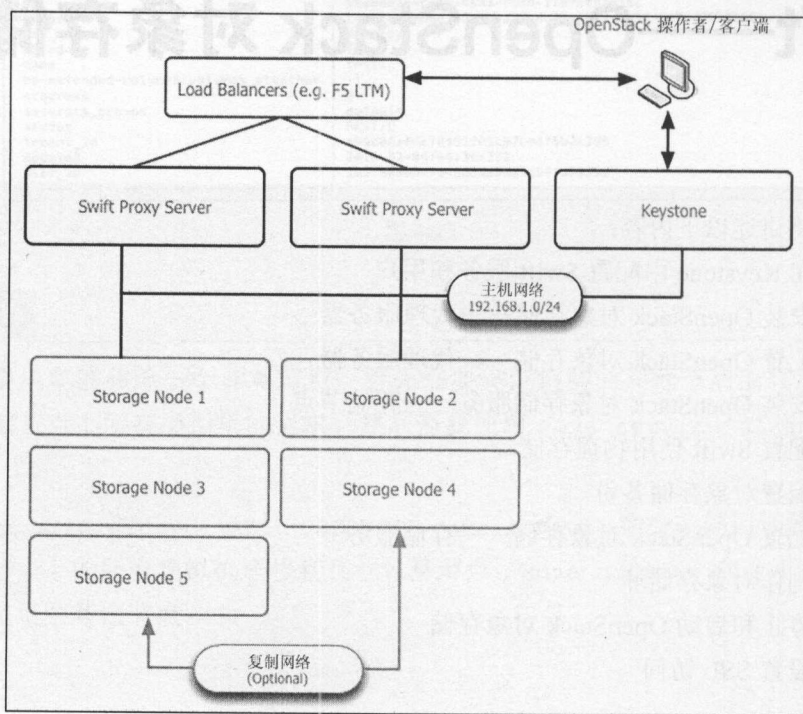


图 5-1



访问 <http://bit.ly/OpenStackCookbookSwift>，了解本章相关的多节点 Swift 安装细节。

5.2 在 Keystone 中配置 Swift 服务和用户

在 Keystone 中配置 OpenStack 对象存储环境，包括定义服务和端点，并在 tenant 服务下创建相应的用户。本章后面配置存储服务时会用到这些细节。

在这个环境下，定义代理服务器的地址和端口。在测试环境下，代理服务器的 IP 地址是 192.168.100.209（公共 IP）和 172.16.0.209（内部/管理）。在生产环境下，会是由负载均衡管理的地址。

准备工作

确保登录到了控制节点，或者相应的客户端可以访问用来配置 keystone 的控制节点。如果使用 vagrant 创建了该节点，可使用如下命令访问。

```
vagrant ssh controller
```

操作步骤

执行如下步骤，在 Keystone 中配置 Swift 服务。

1. 使用 Keystone 客户端，设置如下环境变量，配置为使用管理员账号。

```
export ENDPOINT=192.168.100.209
export SERVICE_TOKEN=ADMIN
export SERVICE_ENDPOINT=https://${ENDPOINT}:35357/v2.0
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

2. 按如下方式在 Keystone 中定义 swift 服务。

```
# Configure the OpenStack Object Storage Endpoint
keystone service-create \
  --name swift \
  --type object-store \
  --description 'OpenStack Object Storage Service'
```

3. 接下来定义端点。这里，将公共端点设置为公共网络 192.168.100.0/24，将内部和管理 URL 设置到管理网络 172.16.0.0/16。

```
# Service Endpoint URLs
SWIFT_SERVICE_ID=$(keystone service-list \
  | awk '/\ swift\ / {print $2}')
```

```
PUBLIC_URL="http://192.168.100.209:8080/v1/AUTH_${tenant_id}s"
ADMIN_URL="http://172.16.0.209:8080/v1"
INTERNAL_URL="http://172.16.0.209:8080/v1/AUTH_${tenant_id}s"
```

```
keystone endpoint-create --region RegionOne \
  --service_id $SWIFT_SERVICE_ID \
  --publicurl $PUBLIC_URL \
  --adminurl $ADMIN_URL \
  --internalurl $INTERNAL_URL
```

4. 端点配置为指向 OpenStack 对象服务器后，可以设置 swift 用户，这样代理服务器就可以通过 OpenStack 身份认证服务器进行认证了。

```
# Get the service tenant ID
SERVICE_TENANT_ID=$(keystone tenant-list \
  | awk '/\ service\ / {print $2}')
```



```
# Create the swift user with password swift
keystone user-create \
  --name swift \
  --pass swift \
  --tenant_id $SERVICE_TENANT_ID \
  --email swift@localhost \
  --enabled true
# Get the swift user id
USER_ID=$(keystone user-list \
  | awk '/\ swift\ / {print $2}')

# Get the admin role id
ROLE_ID=$(keystone role-list \
  | awk '/\ admin\ / {print $2}')

# Assign the swift user admin role in service tenant
keystone user-role-add \
  --user $USER_ID \
  --role $ROLE_ID \
  --tenant_id $SERVICE_TENANT_ID
```

工作原理

使用 Swift 时，我们将通过 Keystone 验证用户，因此需要在 Keystone 中添加 Swift 设置才能正常工作。首先，定义 Swift 服务。在本例中，Swift 的类型是 object-store。然后，定义端点。Swift 将利用两个网络：一个标记为 public 的面向用户的网络（指的是客户端 API 请求经过的网络），以及一个用于服务间通信的内部网络。最后，创建 service 租户用户。在本例中，使用 swift 作为用户名，密码也设置为 swift。在生产环境下，应该选择强度和随机性更高的密码。和其他 OpenStack 服务一样，该用户被赋予了 service 租户下的管理员角色。

5.3 安装 OpenStack 对象存储服务——代理服务器

客户端通过 Swift 代理服务器连接 OpenStack 对象存储。这可以在必要时扩容 OpenStack 对象存储环境，同时不影响客户端连接的前端。代理服务器上要安装如下软件包。

- ❑ swift：在其他 OpenStack 对象存储包中共享的底层通用文件，包括 swift 客户端。
- ❑ swift-proxy：负责提供 OpenStack 存储节点访问功能的代理服务。
- ❑ memcached：高性能内存对象缓存系统
- ❑ python-swiftclient：Swift 客户端，通过命令行界面（CLI）访问 OpenStack 对象存储环境。
- ❑ python-keystoneclient：支持服务于 Keystone 通信的客户端及库。
- ❑ python-webob：提供 WSGI（Web Service Gateway Interface）请求和响应对象的

Python 模块。

❑ curl: 访问 Web 资源的命令行工具。

Swift 会用到多台代理服务器。事实上,在生产环境中至少需要两台,并放置在负载均衡之后。在环境中的所有代理服务器上重复本节的操作步骤。

准备工作

确保登录到了 swift-proxy 节点。如果使用 vagrant 创建了该节点,可使用如下命令访问。

```
vagrant ssh swift-proxy
```



确保 OpenStack 对象存储环境中的所有主机已经安装了 NTP。访问 <http://bit.ly/OpenStackCookbookPreReqs>, 了解相关操作指南。

操作步骤

在 Ubuntu 14.04 上,只需要使用 apt-get 工具即可安装 OpenStack 代理服务器和相关软件包,因为 Ubuntu 官方资源库提供了这些安装包。执行如下步骤,在每个存储节点上安装所需的软件包。

1. 按照如下步骤,在存储节点上安装所有 OpenStack 对象存储软件包。

```
sudo apt-get update
sudo apt-get install swift swift-proxy memcached \
python-keystoneclient python-swiftclient \
curl python-webob
```

2. 然后创建 Swift 将使用的目录,并设置合适的权限。

```
# Create signing directory and set owner to swift
mkdir /var/swift-signing
chown -R swift /var/swift-signing
```

```
# Create cache directory and set owner to swift
mkdir -p /var/cache/swift
chown -R swift:swift /var/cache/swift
```

```
# Create config directory and set owner to swift
mkdir -p /etc/swift
chown -R swift:swift /etc/swift
```



在环境中的每个存储节点重复该安装操作。

工作原理

从 Ubuntu 软件包资源库里安装 OpenStack 对象存储非常直接,可以用很容易理解的方法

式获取 OpenStack 到 Ubuntu 服务器上。并且在稳定性和升级路径上增添了更大的确定性，不会偏离主干分支。

5.4 配置 OpenStack 对象存储服务——代理服务器

客户端通过代理服务器连接 OpenStack 对象存储。这可以在必要时扩容 OpenStack 对象存储环境，同时不影响客户端连接的前端。只需要编辑/etc/swift/proxy-server.conf 文件，即可配置 Swift 代理服务。

准备工作

确保登录到了 swift-proxy 节点。如果使用 vagrant 创建了该节点，可使用如下命令访问。

```
vagrant ssh swift-proxy
```

操作步骤

执行如下步骤，配置 OpenStack 对象存储的代理服务器。

1. 首先，创建/etc/swift/proxy-server.conf 文件，加入如下内容。

```
[DEFAULT]
bind_port = 8080
user = swift
swift_dir = /etc/swift
log_level = DEBUG

[pipeline:main]
# Order of execution of modules defined as follows
pipeline = catch_errors healthcheck cache authToken
keystone proxy-server

[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true
account_autocreate = true
set log_name = swift-proxy
set log_facility = LOG_LOCAL0
set log_level = INFO
set access_log_name = swift-proxy
set access_log_facility = SYSLOG
set access_log_level = INFO
set log_headers = True

[filter:healthcheck]
use = egg:swift#healthcheck

[filter:catch_errors]
```



```

use = egg:swift#catch_errors

[filter:cache]
use = egg:swift#memcache
set log_name = cache

[filter:authtoken]
paste.filter_factory =
keystoneclient.middleware.auth_token:filter_factory

# Delaying the auth decision is required to support tokenless
# usage for anonymous referrers ('.r:*').
delay_auth_decision = true

# auth_* settings refer to the Keystone server
auth_uri = https://192.168.100.200:35357/v2.0/
identity_uri = https://192.168.100.200:5000
insecure = True      # using self-signed certs

# the service tenant and swift username and password
created in Keystone
admin_tenant_name = service
admin_user = swift
admin_password = swift

signing_dir = /var/swift-signing

[filter:keystone]
use = egg:swift#keystoneauth
operator_roles = admin, Member

```

2. 再创建一个新文件/etc/swift/swift.conf，环境中的所有服务器上都必须有该文件。将内容相同的文件拷贝至所有服务器（代理服务器及存储节点）。

```

[swift-hash]
# Random unique string used on all nodes
swift_hash_path_prefix=a4rUmUIgJYXpKhbh
swift_hash_path_suffix=NESuuUEqc6OXwy6X

```

工作原理

/etc/swift/proxy-server.conf 文件的内容定义了 OpenStack 对象存储的代理服务如何配置。

对于本书来说，我们将以用户 swift 的身份在 8080 端口上运行代理，并使用 INFO 日志级别记录日志到 syslog（INFO 是默认的日志级别）。

[filter:authtoken] 和 [filter:keystone] 部分将 OpenStack 对象存储代理连接至运行在控制节点虚拟机上的 keystone 服务。[filter:authtoken] 下的内容，与配置 keystone 时其他服务的语法相同。

/etc/swift/swift.conf 文件不是只存在于代理服务器。该文件必须放置在所有 Swift 服务器上，而且内容要完全相同。在存储节点上配置时，会复制该文件。

延伸阅读

- ❑ 安装 OpenStack Swift 服务时，会自动安装下面这个文件，其中介绍了更多复杂的选项和功能：/usr/share/doc/swift-proxy/proxy-server.conf-sample。

5.5 安装 OpenStack 对象存储服务——存储节点

存储节点运行了一些 OpenStack 对象存储服务。这些服务和库可以使用 apt 安装，具体如下。

- ❑ swift：在其他 OpenStack 对象存储包中共享的底层通用文件，包括 swift 客户端。
- ❑ swift-account：访问 OpenStack 存储的账号服务。
- ❑ swift-object：负责对象存储和编排同步的包。
- ❑ swift-container：OpenStack 对象存储容器服务器。
- ❑ rsyncd：文件备份守护程序，用于跨节点备份对象。
- ❑ python-keystoneclient：支持服务于 Keystone 通信的客户端及库。
- ❑ python-webob：提供 WSGI 请求和响应对象的 Python 模块。

准备工作

确保已经登录到 swift 存储节点。如果使用 vagrant 创建了这些节点，接下来将在这五个存储节点上执行操作。可执行如下命令访问每个节点。

```
vagrant ssh swift-01
vagrant ssh swift-02
vagrant ssh swift-03
vagrant ssh swift-04
vagrant ssh swift-05
```



确保 OpenStack 对象存储环境中的所有主机已经安装了 NTP。访问 <http://bit.ly/OpenStackCookbookPreReqs>，了解相关操作指南。

操作步骤

在 Ubuntu 14.04 上安装 OpenStack 对象存储节点服务非常简单，只需使用 apt-get

工具即可从 Ubuntu 官方资源库中获取。执行如下步骤，在每个存储节点上安装所需的软件包。

1. 按照如下命令安装 OpenStack 对象存储软件包。

```
sudo apt-get update
sudo apt-get install swift swift-account \
    swift-container swift-object python-webob \
    python-keystoneclient rsync
```

2. 然后创建 Swift 会用到的目录，运行如下命令设置合适的权限。

```
# Create signing directory and set owner to swift
mkdir /var/swift-signing
chown -R swift /var/swift-signing
# Create cache directory & set owner to swift
mkdir -p /var/cache/swift
chown -R swift:swift /var/cache/swift

# Create config directory and set owner to swift
mkdir -p /etc/swift
chown -R swift:swift /etc/swift
```



在环境中的每个存储节点重复该安装操作。

工作原理

从 Ubuntu 软件包资源库里安装 OpenStack 对象存储非常直接，可以用很容易理解的方式获取 OpenStack 到 Ubuntu 服务器上。并且在稳定性和升级路径上增添了更大的确定性，不会偏离主干分支。

5.6 配置 Swift 使用物理存储

OpenStack 对象存储 Swift 的架构相对简单。它在前端使用代理服务器将数据传送到分配的存储节点。存储节点可使用任意数量的磁盘。在本节中，配套的虚拟环境由五个存储节点组成，每个都有一个额外磁盘分区/dev/sdb1，挂载在/srv/node/sdb1，也用于写入数据。本节将介绍正确配置这些磁盘卷的过程。

准备工作

确保已经登录到 swift 存储节点。如果使用 vagrant 创建了这些节点，接下来将在这

五个存储节点上执行操作。可执行如下命令访问每个节点。

```
vagrant ssh swift-01
vagrant ssh swift-02
vagrant ssh swift-03
vagrant ssh swift-04
vagrant ssh swift-05
```

操作步骤

在五个存储节点上执行如下步骤，配置 Swift 使用 OpenStack 对象存储磁盘。重复这些步骤，直到每个节点都配置正确。

1. 首先，确存储节点上已经安装了准备磁盘所需的工具。通常都安装了 parted 工具，但默认没有安装 xfs。执行如下命令安装这两个工具。

```
sudo apt-get update
sudo apt-get install parted xfsprogs
```

2. 然后，使用 parted 工具准备额外的磁盘，在 Linux 系统下的名称为/dev/sdb。执行如下命令，创建使用整个磁盘的分区。

```
sudo parted /dev/sdb mklabel msdos
NUM_CYLINDERS=$(sudo parted /dev/sdb unit cyl print \
    | awk '/Disk.*cyl/ {print $3}')
sudo parted mkpart primary 0cyl $NUM_CYLINDERS
```

3. 运行 partprobe 重新读取磁盘分区信息，即可在不重启的情况下使 Linux 系统发现新的分区。

```
sudo partprobe
```

4. 完成后，可以在/dev/sdb1 分区创建文件系统。为此，我们使用 xfs 文件系统工具。

```
sudo mkfs.xfs -i size=1024 /dev/sdb1
```

5. 现在创建要求的挂载点，设置 fstab 挂载新的磁盘区。

```
sudo mkdir /srv/node/sdb1
```

6. 然后编辑/etc/fstab，加入如下内容。

```
/dev/sdb1 /srv/node/sdb1 xfs
    noatime,nodiratime,nobarrier,logbufs=8 0 0
```

7. 现在可以挂载该区。

```
sudo mount /srv/node/sdb1
```

8. 确保 Swift 用户和用户组可以写入数据至该区。

```
sudo chown -R swift:swift /srv/node
```



在环境中的每个存储节点重复该安装操作。

工作原理

首先使用 `parted` 在额外的磁盘上创建了一个新分区，并格式化为 XFS 文件系统。XFS 在处理大对象方面是非常棒的，并且有对象文件系统中所需要的扩展属性（`xattr`）。第一步的命令让 Swift 使用整个磁盘，新分区从柱面 0 一直延续到磁盘的最后一个柱面。



所有文件系统均可用于 OpenStack 对象存储，但是必须支持 `xattr`，该属性在 XFS 中使用最广，支持度最高。

创建完毕后，挂载该区为 `/srv/node/{device_name}`。这个推荐结构方便 OpenStack 对象存储管理员理解 Swift 使用了哪些磁盘。

为了满足 OpenStack 对象存储使用的元数据需求，将 `inode` 大小增加为 1024，在格式化时加入 `-i size=1024` 参数设置。

挂载时设置了其他性能选项。用户不需要记录文件访问时间（`noatime`）和目录访问时间（`nodirtime`）。还支持在合适的时间写入 `write-back` 缓存到磁盘。禁用该特性会提升性能，因为 OpenStack 的高可用特性允许磁盘卷发生故障（因此允许数据写入错误），因此可以禁用这个文件系统的安全措施（通过 `nobarrier` 选项）来提升速度。

5.7 配置对象存储备份

对于一个高度冗余且可扩展的对象存储系统，备份（`replication`）是一个关键要求。Rsync 负责 OpenStack 对象存储环境中对象的备份，必须正确配置才能保证 Swift 正常运行。Rsync 配置完成后，Swift 使用的 3 个服务（账号服务器、容器服务器和对象服务器）设置为 3 个 Rsync 模块。账号服务器提供用户信息，容器服务器提供用户拥有的容器的信息，对象服务器负责存储在容器中的数据。

准备工作

确保已经登录到 swift 存储节点。如果使用 `vagrant` 创建了这些节点，接下来将在这五个存储节点上执行操作。可执行如下命令访问每个节点。

```
vagrant ssh swift-01
```

```
vagrant ssh swift-02
vagrant ssh swift-03
vagrant ssh swift-04
vagrant ssh swift-05
```

操作步骤

在 OpenStack 对象存储中配置备份意味着在每个存储节点上配置 rsync 服务。以下步骤将为每个 OpenStack 对象存储服务（包括账户服务器、容器服务器和对象服务器）设置 Rsync 同步模块。

1. 首先，创建/etc/rsyncd.conf 文件，如下。

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 172.16.0.221 # Amend for each storage node's management
# IP address
```

```
[account]
max connections = 25
path = /srv/node/
read only = false
lock file = /var/lock/account.lock
```

```
[container]
max connections = 25
path = /srv/node/
read only = false
lock file = /var/lock/container.lock
```

```
[object]
max connections = 25
path = /srv/node/
read only = false
lock file = /var/lock/object.lock
```

2. 由于 rsync 进程会以 swift 用户和用户组的身份读写文件，要确保使用的路径 (/srv/node) 和这里的目录都为 swift:swift 用户组所有。

```
sudo chown -R swift:swift /srv/node
```

3. 完成之后，启用 rsync 服务。

```
sudo sed -i 's/=false/=true/' /etc/default/rsync
sudo service rsync start
```



在环境中的每个存储节点重复这些步骤。

工作原理

本节介绍了如何配置 Swift 使用的 `rsyncd.conf`。我们配置了多个 `rsync` 模块作为 Rsync 服务器的目标。另外，`rsyncd.conf` 文件的每个部分都有一些配置指令，例如最大连接数、只读和锁文件。其中大部分的值是不言自明的，需要特别注意的是最大连接数的值。在测试环境中，该值设置的较小，不至于让运行 Swift 的小服务器负载过高。在生产环境中，应该按照 Rsync 文档中的指南调整最大连接数的最优值。但关于如何设置最大连接数这个话题的讨论已经超出了本书的讨论范畴。

5.8 配置 OpenStack 对象存储——存储服务

账户服务器列出节点中可以使用的容器。容器服务器包含了 OpenStack 对象存储环境中可见的对象服务器。对象服务器则包含有 OpenStack 对象存储环境中可见的实际对象。以下步骤必须在环境中所有的存储节点上执行一遍。

准备工作

确保已经登录到 swift 存储节点。如果使用 `vagrant` 创建了这些节点，接下来将在这五个存储节点上执行操作。可执行如下命令访问每个节点。

```
vagrant ssh swift-01
vagrant ssh swift-02
vagrant ssh swift-03
vagrant ssh swift-04
vagrant ssh swift-05
```

操作步骤

本节将创建 4 个不同的账户服务器的配置文件，不同之处在于服务运行的端口和服务对应对应的端口上的单个磁盘路径。按照如下步骤，创建配置文件。

1. 首先，为第一个节点创建一个账户服务器配置文件。编辑 `/etc/swift/account-server.conf`，内容如下。

```
[DEFAULT]
devices = /srv/node
bind_port = 6002
user = swift
log_facility = LOG_LOCAL2

[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account
```

```
[account-replicator]
vm_test_mode = yes
```

```
[account-auditor]
```

```
[account-reaper]
```

2. 然后, 编辑相同节点的容器服务器配置文件。编辑/etc/swift/container-server.conf 文件, 加入以下内容。

```
[DEFAULT]
devices = /srv/node
mount_check = false
bind_port = 6001
user = swift
log_facility = LOG_LOCAL2
```

```
[pipeline:main]
pipeline = container-server
```

```
[app:container-server]
use = egg:swift#container
```

```
[account-replicator]
vm_test_mode = yes
```

```
[account-updater]
```

```
[account-auditor]
```

```
[account-sync]
```

```
[container-auditor]
```

```
[container-replicator]
```

```
[container-updater]
```

3. 接下来配置最后一个服务, 对象服务器。编辑/etc/swift/object-server.conf 文件, 加入如下内容。

```
[DEFAULT]
devices = /srv/node
mount_check = false
bind_port = 6000
user = swift
log_facility = LOG_LOCAL2
```

```
[pipeline:main]
pipeline = object-server
```

```
[app:object-server]
use = egg:swift#object
```

```
[object-replicator]
```

```
vm_test_mode = yes
```

```
[object-updater]
```

```
[object-auditor]
```

4. 创建 Swift 配置文件/etc/swift/swift.conf，内容与 5.4 节的第二步中列出的相同。

```
[swift-hash]
# Random unique string used on all nodes
swift_hash_path_prefix=a4rUmUIgJYXpKhbh
swift_hash_path_suffix=NESuuUEqc6OXwy6X
```



在环境中的每个存储节点重复这些步骤。

工作原理

本节配置的是每个存储节点上运行的 3 个服务：账户服务器、容器服务器和对象服务器。每个服务器在存储节点上运行的端口不同，通过 bind_port 标记定义。

```
account-server: bind_port = 6000
container-server: bind_port = 6001
object-server: bind_port = 6002
```

然后在创建 OpenStack 对象存储相关联的存储环时指向这些端口（地址为存储节点的地址）。

它们指向的是设备挂载目录的父目录。在本例中，devices 标记指向的是/srv/node，因为磁盘挂载在/srv/node/sdb1。

5.9 制作对象存储环

最后一步是创建对象环（ring）、账户环和容器环。每个虚拟节点就位于这些环中。OpenStack 对象存储环负责跟踪数据在集群中的存储位置。OpenStack 对象存储认可三种环：账户环、容器环和对象环。为了方便快速重建集群中的环，我们将创建一个脚本，用来执行必要的重建步骤。

准备工作

确保登录到了 swift-proxy 节点，并且安装和配置了运行 Swift 所需的软件包。另外，要按照本章前面的描述安装并配置五个存储节点。如果使用 vagrant 创建了该节点，可使用如下命令访问。

```
vagrant ssh swift-proxy
```


操作步骤

执行如下步骤，创建 OpenStack 对象存储服务使用的 3 种环。

1. 编写脚本，是创建 OpenStack 对象环境所需环的最便捷方法。创建 `/usr/local/bin/remakerings`，加入如下内容。



可从 <http://bit.ly/OpenStackCookbookSwift> 下载该文件。

```
#!/bin/bash
```

```
cd /etc/swift
```

```
rm -f *.builder *.ring.gz backups/*.builder
backups/*.ring.gz
```

```
# Object Ring
```

```
swift-ring-builder object.builder create 18 3 1
```

```
swift-ring-builder object.builder add rlz1-
172.16.0.221:6000/sdb1 1
```

```
swift-ring-builder object.builder add rlz1-
172.16.0.222:6000/sdb1 1
```

```
swift-ring-builder object.builder add rlz1-
172.16.0.223:6000/sdb1 1
```

```
swift-ring-builder object.builder add rlz1-
172.16.0.224:6000/sdb1 1
```

```
swift-ring-builder object.builder add rlz1-
172.16.0.225:6000/sdb1 1
```

```
swift-ring-builder object.builder rebalance
```

```
# Container Ring
```

```
swift-ring-builder container.builder create 18 3 1
```

```
swift-ring-builder container.builder add rlz1-
172.16.0.221:6001/sdb1 1
```

```
swift-ring-builder container.builder add rlz1-
172.16.0.222:6001/sdb1 1
```

```
swift-ring-builder container.builder add rlz1-
172.16.0.223:6001/sdb1 1
```

```
swift-ring-builder container.builder add rlz1-
172.16.0.224:6001/sdb1 1
```

```
swift-ring-builder container.builder add rlz1-
172.16.0.225:6001/sdb1 1
```

```
swift-ring-builder container.builder rebalance
```

```
# Account Ring
```

```
swift-ring-builder account.builder create 18 3 1
```

```
swift-ring-builder account.builder add rlz1-
172.16.0.221:6002/sdb1 1
```

```
swift-ring-builder account.builder add rlz1-
172.16.0.222:6002/sdb1 1
```

```

swift-ring-builder account.builder add rlz1-
172.16.0.223:6002/sdb1 1
swift-ring-builder account.builder add rlz1-
172.16.0.224:6002/sdb1 1
swift-ring-builder account.builder add rlz1-
172.16.0.225:6002/sdb1 1
swift-ring-builder account.builder rebalance

```

2. 现在, 运行该脚本:

```

sudo chmod +x /usr/local/bin/remakerings
sudo /usr/local/bin/remakerings

```

输出如图 5-2 所示。

```

Device d0rlz1-172.16.0.221:6000R172.16.0.221:6000/sdb1_"" with 1.0 weight got id 0
Device d1rlz1-172.16.0.222:6000R172.16.0.222:6000/sdb1_"" with 1.0 weight got id 1
Device d2rlz1-172.16.0.223:6000R172.16.0.223:6000/sdb1_"" with 1.0 weight got id 2
Device d3rlz1-172.16.0.224:6000R172.16.0.224:6000/sdb1_"" with 1.0 weight got id 3
Device d4rlz1-172.16.0.225:6000R172.16.0.225:6000/sdb1_"" with 1.0 weight got id 4
Reassigned 262144 (100.00%) partitions. Balance is now 0.00.
Device d0rlz1-172.16.0.221:6001R172.16.0.221:6001/sdb1_"" with 1.0 weight got id 0
Device d1rlz1-172.16.0.222:6001R172.16.0.222:6001/sdb1_"" with 1.0 weight got id 1
Device d2rlz1-172.16.0.223:6001R172.16.0.223:6001/sdb1_"" with 1.0 weight got id 2
Device d3rlz1-172.16.0.224:6001R172.16.0.224:6001/sdb1_"" with 1.0 weight got id 3
Device d4rlz1-172.16.0.225:6001R172.16.0.225:6001/sdb1_"" with 1.0 weight got id 4
Reassigned 262144 (100.00%) partitions. Balance is now 0.00.
Device d0rlz1-172.16.0.221:6002R172.16.0.221:6002/sdb1_"" with 1.0 weight got id 0
Device d1rlz1-172.16.0.222:6002R172.16.0.222:6002/sdb1_"" with 1.0 weight got id 1
Device d2rlz1-172.16.0.223:6002R172.16.0.223:6002/sdb1_"" with 1.0 weight got id 2
Device d3rlz1-172.16.0.224:6002R172.16.0.224:6002/sdb1_"" with 1.0 weight got id 3
Device d4rlz1-172.16.0.225:6002R172.16.0.225:6002/sdb1_"" with 1.0 weight got id 4
Reassigned 262144 (100.00%) partitions. Balance is now 0.00.

```

图 5-2

3. 上一步完成后 (需要一些时间), 会在/etc/swift 目录下创建 3 个压缩文件, 分别为/etc/swift/account.ring.gz, /etc/swift/container.ring.gz, /etc/swift/object.ring.gz。这些文件需要放入所有存储节点的/etc/swift 目录下。

从代理服务器的/etc/swift/ 目录复制所有的*.gz 文件到每个存储节点的/etc/swift 目录。

工作原理

在 Swift 中, 环的功能是跟踪数据在指定 Swift 集群中的存储位置。在本例中, 我们提供了创建环的细节, 并重建了相关的环。

使用 swift-ring-builder 命令可创建环, 涉及如下步骤, 每个环类型 (对象、容器和账户) 都会重复这些步骤。

1. 使用如下语法创建环:

```

swift-ring-builder builder_file create \
part_power replicas min_part_hours

```

语法指定一个 builder_file 创建 3 个参数, part_power、replicas 和 min_part_hours。这意味着 $2^{\text{part_power}}$ (本例设为 18) 是要创建的分区数, replicas 是环里数

据的备份数（本例设为3），min_part_hours（本例设为1）是特定分区可一次迁移的间隔小时数。

2. 使用如下语法指派设备到环：

```
swift-ring-builder builder_file add \
    zzone-ip:port/device_name weight
```

添加节点到环时要使用第一步中创建的 builder_file。然后指定设备所在的区（如1，前缀为z）；ip（172.16.0.222）是设备所在服务器的IP地址，port（如6000）是服务器运行的端口号，device_name是服务器上设备的名称（如sdbl）。weight是浮点数，决定了设备相对集群中的其他设备有多少个分区。

3. 一个均衡的 Swift 环中，各个节点在提供配置的备份数时，数据交换量可以做到最低。第6章和第7章会提供一些重新均衡 Swift 环的案例。在/etc/swift目录下使用如下语法，即可对环进行再均衡。

```
swift-ring-builder builder_file rebalance
```

上述命令会在环中的磁盘卷中分布分区。对每种环（对象、容器和账户）分别执行上述步骤。

完成 swift-ring-builder 步骤后，记得复制产生的 account.ring.gz、container.ring.gz 和 object.ring.gz 文件到环境中的每个节点，包括可能会配置的其他代理服务器。

5.10 停止和启动 OpenStack 对象存储

在所有节点上安装 OpenStack 对象存储服务后，可以启动用于存储对象和镜像的服务了。

准备工作

确保登录到了所有节点，并安装、配置了运行 Swift 所需的软件包。如果使用 vagrant 创建了该环节，可执行如下命令访问所有节点。

```
vagrant ssh swift-proxy
vagrant ssh swift-01
vagrant ssh swift-02
vagrant ssh swift-03
vagrant ssh swift-04
vagrant ssh swift-05
```

操作步骤

可使用名叫 swift-init 的工具控制 OpenStack 对象存储服务。使用该工具，可启动、停止和重启节点上的不同服务。

在对象存储节点上，通过如下命令启动、停止和重启所有服务。

```
sudo swift-init all start
sudo swift-init all stop
sudo swift-init all restart
```

在代理服务器节点上，通过如下命令启动、停止和重启代理服务。

```
sudo swift-init proxy-server start
sudo swift-init proxy-server stop
sudo swift-init proxy-server restart
```

在所有节点上，确保使用 start 命令启动了所有服务。

工作原理

使用如下语法，即可启动、停止和重启 OpenStack 对象存储的相关服务。

```
sudo swift-init all {start, stop, restart}
sudo swift-init swift-proxy {start, stop, restart}
```

5.11 配置 SSL 访问

设置 SSL 访问是为了在客户端和 OpenStack 对象存储环境之间提供安全的链接，就像 SSL 在其他任意的 Web 服务中所提供的安全访问一样。为此，需要配置代理服务器使用 SSL 证书。



在生产环境中，不会直接在代理服务器上配置 SSL。建议使用硬件负载均衡或其他合适的设备完成 SSL 卸载。本节中介绍的 SSL 配置方式仅用于测试和开发。

准备工作

确保登录到了 swift-proxy 节点，并安装、配置了运行 Swift 所需的软件包。如果使用 vagrant 创建了该环节，可执行如下命令访问节点。

```
vagrant ssh swift-proxy
```

操作步骤

配置 OpenStack 对象存储在客户端和代理服务器直接建立安全链接，步骤如下。

1. 为了提供到代理服务器的 SSL 访问，需要创建证书，如下。

```
cd /etc/swift
sudo openssl req -new -x509 -nodes -out cert.crt \
-keyout cert.key
```

2. 接下来, 需要回答认证过程中的几个问题, 如图 5-3 所示。

3. 一旦创建完毕, 就通过编辑 `/etc/swift/proxy-server.conf` 文件配置代理服务器使用认证证书和密钥。

```
bind_port = 443
cert_file = /etc/swift/cert.crt
key_file = /etc/swift/cert.key
```

```
Generating a 2048 bit RSA private key
.....+++
writing new private key to 'cert.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:GB
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Cookbook
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:192.168.100.210
Email Address []:
```

图 5-3

4. 到这里, 可以使用 `swift-init` 命令重新启动代理服务器, 应用更改。

```
sudo swift-init proxy-server restart
```

5. 现在需要更新 Keystone 端点反映更改。首先移除当前项, 然后添加修改后的细节到端点。先设置环境变量, 确保拥有管理权限, 或者执行如下操作。

```
export ENDPOINT=192.168.100.200
export SERVICE_TOKEN=ADMIN
export SERVICE_ENDPOINT=https://${ENDPOINT}:35357/v2.0
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

6. 使用如下命令列出所有端点, 验证需要删除的项。

```
keystone endpoint-list
```

该命令的输出如图 5-4 所示。Swift 端点已经高亮, 缩减了输出内容以适合页面。

id	region	publicurl
05e3619f45a24f8ba92cf49a6c56f222	RegionOne	http://172.16.0.211:8776/v1/(tenant_id)s
144c754040a14ce29024298d9170956b	RegionOne	http://192.168.100.200:9696
3482d9ccfe2241699be53cdf2de86f28	RegionOne	https://192.168.100.200:5000/v2.0
7807ae8bf89e4b429012b095399bf13c	RegionOne	http://192.168.100.200:8773/services/Cloud
bd46a06576a3489ba9f9a8a7eaa2b2bd	RegionOne	http://swift-proxy:8080/v1/AUTH_(tenant_id)s
caacafe99f304cc5a0ab4f786f25c048	RegionOne	http://192.168.100.200:9292/v2
d14fafa3f74f4a1fa77b91eb2d482d2c	RegionOne	http://192.168.100.200:8774/v2/(tenant_id)s

图 5-4

7. 执行如下命令，移除 Swift 端点。

```
keystone endpoint-delete bd46a06576a3489ba9f9a8a7eaa2b2bd
```

8. 然后添加带有新值的正确端点。

```
PUBLIC_URL="https://192.168.100.209:443/v1/AUTH_\$(tenant_id)s"
```

```
ADMIN_URL="https://172.16.0.209:443/v1"
```

```
INTERNAL_URL=="https://172.16.0.209:443/v1/AUTH_\$(tenant_id)s"
```

```
keystone endpoint-create --region RegionOne \
    --service_id $SWIFT_SERVICE_ID \
    --publicurl $PUBLIC_URL \
    --adminurl $ADMIN_URL \
    --internalurl $INTERNAL_URL
```

工作原理

配置 OpenStack 对象存储服务使用 SSL，还要配置相关的代理服务器使用 SSL。首先使用 `openssl` 命令配置一个自签名的证书，按要求填写不同域的内容。一个重要的域是 **Common Name**。将完全限定会使用连接到的 Swift 服务器的域名（FQDN）主机名或 IP 地址。

一旦完成配置，就需要指定希望代理服务器侦听的端口。由于配置了 SSL HTTPS 连接，因此将使用标准的 TCP 端口，HTTPS 默认使用 443。当发出请求时，指定使用在第一步骤中所创建的证书和密钥，该信息将呈现给最终用户，以允许安全的数据传输。至此，再重新启动代理服务器侦听 443 端口。

最后，修改 Keystone 中的记录匹配之前的修改。为此，要先列出可用端点，确认 Swift 服务的端点 ID；然后使用 `keystone endpoint-delete $ENDPOINT` 命令删除该端点，并加入新的端点，这里要确保指定 https 协议和 443 端口。

完成上述操作之后，我们可以像往常一样使用 Swift，最终用户不需要做任何更改即可看到更改后的效果。

第 6 章

使用 OpenStack 对象存储

本章将讲述以下内容：

- ☐ 安装 swift 客户端工具
- ☐ 创建容器
- ☐ 上传对象
- ☐ 上传大对象
- ☐ 列出容器和对象
- ☐ 下载对象
- ☐ 删除容器和对象
- ☐ 使用 OpenStack 对象存储访问控制列表
- ☐ 两个 Swift 集群间进行容器同步

6.1 简介

现在有了一个 OpenStack 对象存储运行环境，可以用来存储文件了。为此，可以使用 swift 客户端工具。它将帮助用户操作 OpenStack 对象存储环境，包括创建容器、上传文件、检索，并根据需要设置权限。

6.2 安装 swift 客户端工具

为了操作 OpenStack 对象存储环境，需要在客户端安装合适的工具。Swift 模块附带有 swift 工具，可以上传、下载并修改 OpenStack 对象存储环境中的文件。

准备工作

确保登录到了一台 Ubuntu 主机，且该主机能访问位于公共网络 192.168.100.0/24 的 OpenStack 环境。该主机将用于运行 OpenStack 环境的客户端工具。如果使用的是配套的 Vagrant 环境，可以使用控制节点。该节点上安装了 python-swiftclient 软件包，提供了 swift 命令行工具。

如果使用 Vagrant 创建了该节点，可以执行如下命令。

```
vagrant ssh controller
```

确保已经设置了如下证书（如果使用的不是 Vagrant 环境，请视情况调整证书和密钥文件的路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

在联网的机器安装 swift 客户端非常容易，可以方便的通过 apt-get 工具直接从 Ubuntu 资源库下载获取。

1. swift 客户端是通过安装 swift 包和 OpenStack 身份认证服务 Keystone 的 Python 库一起安装的。通过以下命令执行。

```
sudo apt-get update
sudo apt-get install python-swiftclient python-keystone
```

2. 前面的命令将会下载所需的软件包和依赖的 Python 库，无须进一步配置。通过执行以下命令测试 swift 是否安装成功，能否连接到存储服务器。

```
swift stat -v
```

3. 它将返回关于租户 cookbook 的一位用户 admin 访问到的 OpenStack 对象存储环境的相关统计数据。图 6-1 所示为一个示例。

```
StorageURL: http://swift-proxy:8080/v1/AUTH_51e03fdce75e4b088ad1713d95a59e6e
Auth Token: ce6e955daf1d44d9afc353a0ff7d7d86
Account: AUTH_51e03fdce75e4b088ad1713d95a59e6e
Containers: 0
Objects: 0
Bytes: 0
Content-Type: text/plain; charset=utf-8
X-Timestamp: 1435349039.35466
X-Trans-Id: tx7b4dd493da5747bbb6c0c-00558db02f
X-Put-Timestamp: 1435349039.35466
```

图 6-1

工作原理

swift 客户端在 Ubuntu 下安装非常容易, 下载之后无须任何额外的配置, 因为通过命令行与 OpenStack 对象存储通信所需的参数都已安装完毕。

6.3 创建容器

一个容器可以被看做是 OpenStack 对象存储里的 root 文件夹, 用来存储对象。创建对象和容器有多种方式实现, 一种简单的方式就是通过 swift 客户端工具。

准备工作

确保登录到了一台 Ubuntu 主机, 且该主机能访问位于公共网络 192.168.100.0/24 的 OpenStack 环境。该主机将用于运行 OpenStack 环境的客户端工具。如果使用的是配套的 Vagrant 环境, 可以使用控制节点。该节点上安装了 python-swiftclient 软件包, 提供了 swift 命令行工具。

如果使用 Vagrant 创建了该节点, 可以执行如下命令。

```
vagrant ssh controller
```

确保已经设置了如下证书(如果使用的不是 Vagrant 环境, 请视情况调整证书和密钥文件的路径)。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/akey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

执行以下步骤在 OpenStack 对象存储里创建一个容器。

1. 使用 swift 工具, 在 OpenStack 对象存储服务器里创建一个名为 test 的容器, 执行以下命令。

```
swift post test
```

2. 为了验证容器是否创建成功, 可以列出 OpenStack 对象存储环境中所有容器, 执行以下命令。

```
swift list test
```


以上命令将列出 OpenStack 对象存储环境中所有容器。

工作原理

使用 swift 工具创建容器非常简单，语法如下。

```
swift post container_name
```

6.4 上传对象

对象就像是一个容器里存储的文件或目录。上传对象也可以通过多种方式实现。一种简单的方式就是通过 swift 客户端工具，支持创建、删除和修改 OpenStack 环境中的容器和对象。按照本节所述方法，可以上传最大 5GB 的单个对象到 OpenStack 对象存储。

准备工作

确保登录到了一台 Ubuntu 主机，且该主机能访问位于公共网络 192.168.100.0/24 的 OpenStack 环境。该主机将用于运行 OpenStack 环境的客户端工具。如果使用的是配套的 Vagrant 环境，可以使用控制节点。该节点上安装了 python-swiftclient 软件包，提供了 swift 命令行工具。

如果使用 Vagrant 创建了该节点，可以执行如下命令。

```
vagrant ssh controller
```

确保已经设置了如下证书（如果使用的不是 Vagrant 环境，请视情况调整证书和密钥文件的路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

执行以下步骤，在 OpenStack 对象存储里上传对象。

上传文件

通过如下步骤上传文件：

1. 在/tmp 目录下创建一个 500 MB 的文件用于上传，执行以下命令。

```
dd if=/dev/zero of=/tmp/example-500Mb bs=1M count=500
```

2. 上传该文件到 OpenStack 对象存储账号, 执行以下命令。

```
swift upload test /tmp/example-500Mb
```

上传目录及其内容

使用如下步骤上传目录。

1. 创建一个目录以及两个文件, 上传到 OpenStack 对象存储环境中, 执行以下命令。

```
mkdir /tmp/test
dd if=/dev/zero of=/tmp/test/test1 bs=1M count=20
dd if=/dev/zero of=/tmp/test/test2 bs=1M count=20
```

2. 上传目录及其内容, 可以使用相同的命令, 但必须指明目录。该目录下的文件将被递归上传。执行以下命令。

```
swift upload test /tmp/test
```

上传多个对象

可以一次上传多个对象。为此, 需要在命令行中指定每一个对象。例如, 上传 test1 和 test2 文件, 执行以下命令。

```
swift upload test /tmp/test/test1 /tmp/another/test2
```

工作原理

通过 swift 客户端工具上传文件到 OpenStack 对象存储环境非常简便。可以上传单个文件或者上传整个目录, 语法如下。

```
swift upload container_name file | directory {file|directory ... }
```



注意, 当上传文件时, 创建的对象会以用户指定的形式传给 swift 客户端, 包括全路径。例如, 上传 /tmp/example-500Mb 时会以对象 tmp/example-500Mb 形式上传。这是由于 OpenStack 对象存储不是像计算机和台式机那样使用的树形分级文件系统, 路径是由一个斜杠分割而成的 (/或\)。OpenStack 对象存储包含一个容器里保存的对象集合, 斜杠是对象名称的一部分。

6.5 上传大对象

上传到 OpenStack 对象存储的单个对象不得超过 5 GB。但是, 通过把对象拆分为几段,

下载一个单独的对象的大小几乎是没有限制的。大文件分段上传后会创建一个清单文件，下载时会发送一个单独的对象建立所有段的索引。通过把对象分拆成小块，还能通过并行上传提升效率。

准备工作

确保登录到了一台 Ubuntu 主机，且该主机能访问位于公共网络 192.168.100.0/24 的 OpenStack 环境。该主机将用于运行 OpenStack 环境的客户端工具。如果使用的是配套的 Vagrant 环境，可以使用控制节点。该节点上安装了 python-swiftclient 软件包，提供了 swift 命令行工具。

如果使用 Vagrant 创建了该节点，可以执行如下命令。

```
vagrant ssh controller
```

确保已经设置了如下证书（如果使用的不是 Vagrant 环境，请视情况调整证书和密钥文件的路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

执行以下步骤通过拆分成小段来上传大对象。

1. 在 /tmp 目录下创建一个 1 GB 文件作为上传示例文件。执行以下命令。

```
dd if=/dev/zero of=/tmp/example-1Gb bs=1M count=1024
```

2. 使用分段把它拆分成小块（如 100 MB/段）而不是作为一个单独的对象上传整个大文件。为此，通过 -s 选项指定段的大小，执行以下命令。

```
swift upload test -S 102400000 /tmp/example-1Gb
```



注意，分段大小的单位是位。

输出如图 6-2 所示。其中显示了每个上传状态。


```
tmp/example-1Gb segment 7
tmp/example-1Gb segment 5
tmp/example-1Gb segment 2
tmp/example-1Gb segment 0
tmp/example-1Gb segment 3
tmp/example-1Gb segment 6
tmp/example-1Gb segment 4
tmp/example-1Gb segment 10
tmp/example-1Gb segment 1
tmp/example-1Gb segment 8
tmp/example-1Gb segment 9
tmp/example-1Gb
```

图 6-2

工作原理

OpenStack 对象存储非常擅长存储和检索大型对象。为了更加高效,可以拆分大型对象成为小对象,通过维护段间的关系使之仍然当做一个文件看待。这样就可以并行上传大型文件,而不是线性的上传一个单独的文件。为此,使用的语法如下。

```
swift upload container_name -S bytes_to_split large_file
```

现在,当列出账户下的容器时,就有了一个额外的容器,命名为 `test_segments`,它将为文件保持实际的数据片段。`test` 容器仍将该大型对象当作是一个独立的文件。在幕后,这个单一对象的元数据将从 `test_segments` 容器中拉回独立的对象,重构这个大型对象,命令如下。

```
swift list
```

执行前面的命令时,得到以下输出。

```
test
test_segments
```

现在,执行以下命令。

```
swift list test
```

产生以下输出。

```
tmp/example-1Gb
```

还可以列出 `test_segments` 中的内容检查分段,命令如下:

```
swift list test_segments
```

该命令的输出如图 6-3 所示。

```

tmp/example-1Gb/1435350204.204782/1073741824/102400000/00000000
tmp/example-1Gb/1435350204.204782/1073741824/102400000/00000001
tmp/example-1Gb/1435350204.204782/1073741824/102400000/00000002
tmp/example-1Gb/1435350204.204782/1073741824/102400000/00000003
tmp/example-1Gb/1435350204.204782/1073741824/102400000/00000004
tmp/example-1Gb/1435350204.204782/1073741824/102400000/00000005
tmp/example-1Gb/1435350204.204782/1073741824/102400000/00000006
tmp/example-1Gb/1435350204.204782/1073741824/102400000/00000007
tmp/example-1Gb/1435350204.204782/1073741824/102400000/00000008
tmp/example-1Gb/1435350204.204782/1073741824/102400000/00000009
tmp/example-1Gb/1435350204.204782/1073741824/102400000/00000010

```

图 6-3

6.6 列出容器和对象

swift 客户端工具可以轻松列出 OpenStack 对象存储账户中的容器和对象。

准备工作

确保登录到了一台 Ubuntu 主机，且该主机能访问位于公共网络 192.168.100.0/24 的 OpenStack 环境。该主机将用于运行 OpenStack 环境的客户端工具。如果使用的是配套的 Vagrant 环境，可以使用控制节点。该节点上安装了 python-swiftclient 软件包，提供了 swift 命令行工具。

如果使用 Vagrant 创建了该节点，可以执行如下命令。

```
vagrant ssh controller
```

确保已经设置了如下证书（如果使用的不是 Vagrant 环境，请视情况调整证书和密钥文件的路径）。

```

export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/akey.pem
export OS_CACERT=/vagrant/ca.pem

```

操作步骤

执行以下步骤，列出 OpenStack 对象存储环境中的对象。

列出一个容器中的所有对象

在前面的步骤里，上传了一些少量的文件。为了列出 test 容器中的对象，执行以下命令。

```
swift list test
```

输出如图 6-4 所示。

```
tmp/example-500Mb
tmp/test/test1
tmp/test/test2
```

图 6-4

列出一个容器中的指定对象路径

1. 为了只列举出 tmp/test 路径下的文件，需要使用 -p 选项，如下所示。

```
swift list -p tmp/test test
```

这将列举出两个文件，如下所示。

```
tmp/test/test1
tmp/test/test2
```

2. -p 选项支持部分匹配。例如，列举所有以 tmp/ex 开头的文件，可以执行以下命令。

```
swift list -p tmp/ex test
```

这将列出匹配指定字符串的文件。

```
tmp/example-500Mb
```

工作原理

swift 工具是一个基本的但很强大的实用程序，可以帮助用户管理文件；也能以用户希望的方式列举出文件。简单列举容器中的内容的语法如下。

```
swift list {container_name}
```

为了列举一个容器内的特定的路径下的文件，可以为该语法添加 -p 选项。

```
swift list -p path {container_name}
```

6.7 下载对象

既然已经配置了一个具有容器和对象的 OpenStack 对象存储环境，我们也可以使用 swift 客户端下载存储的对象。

准备工作

◆ 确保登录到了一台 Ubuntu 主机，且该主机能访问位于公共网络 192.168.100.0/24 的 OpenStack 环境。该主机将用于运行 OpenStack 环境的客户端工具。如果使用的是配套的

Vagrant 环境，可以使用控制节点。该节点上安装了 `python-swiftclient` 软件包，提供了 `swift` 命令行工具。

如果使用 Vagrant 创建了该节点，可以执行如下命令。

```
vagrant ssh controller
```

确保已经设置了如下证书（如果使用的不是 Vagrant 环境，请视情况调整证书和密钥文件的路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

我们将使用不同的 `swift` 工具选项，从 OpenStack 对象存储环境中下载对象。

下载对象

下载 `tmp/test/test1` 对象，使用以下命令：

```
swift download test tmp/test/test1
```

这将下载对象到文件系统中。由于通过全路径下载文件，该目录结构保存了下来，因此会创建一个新目录 `tmp/test`，并生成一个新的文件 `test1`。

使用 `-o` 选项下载对象

为了不用保存文件结构下载文件，或者只是简单的重命名为其他文件，只需指定 `-o` 选项，如下。

```
swift download test tmp/test/test1 -o test1
```

从一个容器中下载所有对象

也可以下载容器的所有对象到本地文件系统。为此，只需指定所需下载容器，如下：

```
swift download test
```

这将下载 `test` 容器下发现的所有对象。

从我们的 OpenStack 对象存储账号里下载所有对象

还可以下载 OpenStack 对象存储账号下的所有对象。如果有多个容器，所有容器里的

对象都将被下载，可以使用--all 选项，如下：

```
swift download --all
```

这将下载指定容器名内的所有对象，输出如图 6-5 所示。

```
test/tmp/test/test1
test/tmp/test/test2
test/tmp/example-500Mb
```

图 6-5

工作原理

swift 客户端是一个基本但强大的工具，可以帮助用户管理文件。下载对象和容器可以通过以下语法实现。

```
swift download container_name {object ... }
```

下载对象到文件系统并重命名，可使用-o 参数指定一个不同的本地文件名。

```
swift download container_name object -o renamed_object
```

从账户（如所有容器中）下载所有对象，可通过以下语法实现。

```
swift download --all
```

6.8 删除容器和对象

swift 客户端工具还可以直接删除 OpenStack 对象存储中的容器和对象。

准备工作

确保登录到了一台 Ubuntu 主机，且该主机能访问位于公共网络 192.168.100.0/24 的 OpenStack 环境。该主机将用于运行 OpenStack 环境的客户端工具。如果使用的是配套的 Vagrant 环境，可以使用控制节点。该节点上安装了 python-swiftclient 软件包，提供了 swift 命令行工具。

如果使用 Vagrant 创建了该节点，可以执行如下命令。

```
vagrant ssh controller
```

确保已经设置了如下证书（如果使用的不是 Vagrant 环境，请视情况调整证书和密钥文件的路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
```

```
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

我们将使用不同的 swift 工具选项删除 OpenStack 对象存储环境中的对象。

删除对象

要删除对象 tmp/test/test1, 执行以下命令:

```
swift delete test tmp/test/test1
```

这将从 test 容器中删除 tmp/test/test1 对象。

删除多个对象

要删除对象 tmp/test/test2 和 tmp/example-500Mb, 使用以下命令:

```
swift delete test tmp/test/test2 tmp/example-500Mb
```

这将从 test 容器中删除对象 tmp/test/test2 和 tmp/example-500Mb。

删除容器

要删除 test 容器, 使用以下命令。

```
swift delete test
```

删除账户下所有内容

要删除账户下所有容器和对象, 使用以下命令:

```
swift delete --all
```

这将删除所有容器和这些容器下的所有对象。

工作原理

swift 客户端是一个基本但强大的工具, 可以帮助用户管理文件。删除对象和容器可以通过以下语法实现。

```
swift delete {container_name} {object ... }
```

从账户 (如从所有容器中) 删除所有对象, 执行以下语法。

```
swift delete --all
```


6.9 使用 OpenStack 对象存储访问控制列表

访问控制列表 (ACLs)，可以更好地管理对象和容器，且无须要求对特定容器拥有完整的读/写访问权限。通过 ACLs，可以全局地公开容器，或者限制租户和用户访问。

准备工作

确保登录到了一台 Ubuntu 主机，且该主机能访问位于公共网络 192.168.100.0/24 的 OpenStack 环境。该主机将用于运行 OpenStack 环境的客户端工具。如果使用的是配套的 Vagrant 环境，可以使用控制节点。该节点上安装了 python-swiftclient 软件包，提供了 swift 命令行工具。

如果使用 Vagrant 创建了该节点，可以执行如下命令。

```
vagrant ssh controller
```

确保已经设置了如下证书（如果使用的不是 Vagrant 环境，请视情况调整证书和密钥文件的路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

执行以下步骤。

1. 在 OpenStack 身份认证服务器中，首先创建一个账号，只作为 tenant 租户中的一个 Member，名为 user。代码如下。

```
export ENDPOINT=192.168.100.200
export SERVICE_TOKEN=ADMIN
export SERVICE_ENDPOINT=https://${ENDPOINT}:35357/v2.0
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

```
# First get TENANT_ID related to our 'cookbook' tenant
TENANT_ID=$(keystone tenant-list \
    | awk ' / cookbook / {print $2}')
```

```
# We then create the user specifying the TENANT_ID
keystone user-create \
  --name test_user \
  --tenant_id $TENANT_ID \
  --pass openstack \
  --email user@localhost \
  --enabled true

# We get this new user's ID
USER_ID=$(keystone user-list | awk ' / user / {print $2}')

# We get the ID of the 'Member' role
ROLE_ID=$(keystone role-list \
  | awk ' / Member / {print $2}')

# Finally add the user to the 'Member' role in cookbook
keystone user-role-add \
  --user $USER_ID \
  --role $ROLE_ID \
  --tenant_id $TENANT_ID
```

2. 创建好新用户后，现在通过一个拥有管理权限的用户来创建一个容器（因此新创建的用户不能访问该容器），命令如下。

```
swift post testACL
```

3. 然后设置该容器为新用户 test_user 只读，命令如下。

```
swift post -r test_user testACL
```

4. 使用新用户尝试上传一个文件到该容器。

```
swift upload testACL /tmp/test/test1
```

此时将会收到一个“HTTP 403 Forbidden”的消息，如下。

```
Object HEAD failed: https://proxy-server:8080/v1/AUTH_53d87d9b6679
4904aa2c84c17274392b/testACL/tmp/test/test1 403 Forbidden
```

5. 现在，为新用户赋予 testACL 容器的写访问权限。

```
swift post -w test_user -r test_user testACL
```

6. 再次上传该文件，执行成功。

```
swift upload testACL /tmp/test/test1
```

工作原理

授权访问控制是基于容器完成的，可实现用户级别的控制。当用户创建一个容器时，可以将其他用户添加到该容器为其赋予访问权限。这样，这些用户就会获得新创建容器的读写访问权限，示例如下。

```
swift post -w user -r user container
```

6.10 两个 Swift 集群间进行容器同步

从一个 Swift 集群将容器的内容备份到另一个远程容器，对于灾难恢复和主动数据中心的运行来说是一个非常有用的特性，支持用户正常上传对象到某个容器，然后自动将这些内容备份到远程集群的指定容器中。

准备工作

确保登录到了用来备份对象的两台 Swift 代理服务器。该特性的示例可以到 <https://github.com/OpenStackCookbook/VagrantSwift> 下查看。如果使用 Vagrant 环境创建了这些节点，确保已经在运行 swift 和 swift2 集群，并且执行如下命令登录二个节点。

```
vagrant ssh swift
vagrant ssh swift2
```

操作步骤

执行如下步骤，设置容器同步备份。

1. 在两个代理服务器上，编辑/etc/swift/proxy-server.conf 文件，在管道部分添加 container_sync。

```
[pipeline:main]
# Order of execution of modules defined below
pipeline = catch_errors healthcheck cache container_sync authtoken
keystone proxy-server
[filter:container_sync]
use = egg:swift#container_sync
```

2. 在每个代理服务器上，创建/etc/swift/container-sync-realms-conf 文件，写入如下内容。

```
[realm1]
key = realm1key
cluster_swift = http://swift:8080/v1/
cluster_swift2 = http://swift2:8080/v1/
```

3. 在每个代理服务器上，执行如下命令使更改生效。

```
swift-init proxy-server restart
```

4. 在第一个 Swift 集群 (swift) 上，确认第二个集群上 (swift2) 的账户，用于设置集群同步。

```
swift --insecure -V2.0 -A https://swift2:5000/v2.0
-U cookbook:admin
```


-K openstack

上述命令的输出如图 6-6 所示（注意 Account: 行）。

```
Account: AUTH_d81683a9a2dd46cf9cac88c5b8eacala
Containers: 0
Objects: 0
Bytes: 0
Content-Type: text/plain; charset=utf-8
X-Timestamp: 1421140955.13054
X-Trans-Id: tx32e023c22538426096070-0054b4e3db
X-Put-Timestamp: 1421140955.13054
```

图 6-6



注意，该命令中使用了 `--insecure` 标记，因为 Swift2 使用的是自签名的证书，我们没有访问 Swift 节点上生成的证书文件的权限。如果你在各个节点上拷贝了该文件，使其变成可访问的，则可以省略该标记。

5. 在第一个集群 swift 上设置一个名为 `container01` 的容器，与第二个集群 swift2 上名为 `container2` 的容器进行同步。

```
swift -V2.0 -A https://controller:5000/v2.0
-U cookbook:admin -K openstack post
-t '//realm1/swift2/AUTH_d81683a9a2dd46cf9cac88c5b8eacala/
container2'
-k 'myKey' container1
```

6. 在第二个集群上设置上一步中用到的 `container2`，将内容同步回第一个集群上的 `container1` 容器（双向同步）。注意，我们是从 swift 节点上运行下面的命令，而且是远程在 swift2 节点上创建容器。

```
swift --insecure -V2.0 -A https://swift2:5000/v2.0
-U cookbook:admin
-K openstack
post container2
```

7. 上传一个文件到 swift1 上的 `container1`。

```
swift -V2.0 -A https://controller:5000/v2.0
-U cookbook:admin -K openstack
upload container1 my_example_file
```

8. 现在可以查看 swift2 集群上 `container2` 中的内容，会与 swift 集群上 `container1` 中的内容相同。



如果文件还没出现在第二个集群 swift 的 `container2` 上，运行如下命令：`swift-init container-sync once`。

工作原理

对于使用多个数据中心而且容灾方案要求数据在每个数据中心之间保持一致的业务来说，容器同步是一个非常棒的特性。容器同步运行在容器层面，因此可以控制数据同步的目的地。

修改/etc/swift/proxy-server.conf 文件中的管道，通知 Swift 运行容器同步工作即可启用该特性。

配置好后，创建一个名为/etc/swift/container-sync-realms.conf 的文件，写入如下内容：

```
[realm_name]
key = realm_name_key
cluster_name_of_cluster = http://swift1_proxy_server:8080/v1/
cluster_name_of_cluster2 = http://swift2_proxy_server:8080/v1/
```

这个内容很重要，在容器上创建同步服务时会用到，创建同步服务的语法如下：

```
swift post
-t '///realm_name/name_of_cluster2/AUTH_UUID/container_name'
-k 'mykey' container_name_to_be_syncd
```

AUTH_UUID 来自如下命令，会给出远程（接收端）Swift 集群上的用户相关联的账号。

```
swift -V2.0 -A https://cluster2:5000/v2.0
-U tenant:user -K password
stat
```

然后使用密钥，以及/etc/swift/container-sync-realms.conf 文件中设置的密钥，创建一个共享密钥，用于容器间用户验证。

因为这个配置，在向第一个集群上的容器上传文件时，容器被指示同步内容到第二个集群，因此该文件会在后台自动同步过去。

更多参考

容器同步，只是让不同 Swift 节点间备份数据的一种方法。另一个方法是使用全局集群。更多信息，请查看 <https://swiftstack.com/blog/2013/07/02/swift-1-9-0-release/>。

第 7 章

管理 OpenStack 对象存储

本章将讲述以下内容：

- ❑ 用 `swift-init` 管理 OpenStack 对象存储集群
- ❑ 检查集群健康状况
- ❑ 管理 `swift` 集群容量
- ❑ 从集群中删除节点
- ❑ 检测和更换故障硬盘
- ❑ 收集使用情况统计

7.1 简介

OpenStack 对象存储集群日常的管理工作涉及确保集群中的文件复制到正确的节点数量、报告集群使用情况和处理集群的故障。本章在第 6 章的基础之上介绍管理 OpenStack 对象存储需要的工具和流程。

7.2 用 `swift-init` 管理 OpenStack 对象存储集群

OpenStack 对象存储环境中的服务可以使用 `swift-init` 工具管理。该工具允许用户方便地控制 OpenStack 对象存储中所有的守护进程。关于安装和配置 Swift 服务相关的内容，参见第 5 章。

准备工作

登录到任意 OpenStack 对象存储节点。如果使用的是 vagrant 环境，可以通过如下命令访问。

```
vagrant ssh swift-proxy
vagrant ssh swift-01
vagrant ssh swift-02
vagrant ssh swift-03
vagrant ssh swift-04
vagrant ssh swift-05
```

操作步骤

swift-init 工具可以用来控制 OpenStack 对象存储集群中运行的任意守护进程，而不用调用独立的 init 脚本，是一个非常方便的工具。

每个命令可以执行如下。

控制 OpenStack 对象存储代理，执行如下命令。

```
swift-init proxy-server { command }
```

控制 OpenStack 对象存储对象守护程序，执行如下命令。

```
swift-init object { command }
swift-init object-replicator { command }
swift-init object-auditor { command }
swift-init object-updater { command }
```

控制 OpenStack 对象存储容器守护进程，执行如下命令。

```
swift-init container { command }
swift-init container-update { command }
swift-init container-replicator { command }
swift-init container-auditor { command }
```

控制所有 OpenStack 对象存储账户守护进程，使用如下命令。

```
swift-init account { command }
swift-init account-auditor { command }
swift-init account-reaper { command }
swift-init account-replicator { command }
```

控制所有守护进程，使用如下命令。

```
swift-init all { command }
```

{ command } 可以是表 7-1 中的一项。

表 7-1

命令	描述
stop、start 和 restart	命令含义显而易见，start 和 stop 用于启动和停止守护程序对象。Restart 命令为依次执行 stop 和 start 命令的功能。
force-reload 和 reload	做的是同一件事情——正常停止后重启服务器
shutdown	等待当前进程终止后停止服务器
no-daemon	在当前 shell 中启动服务器
no-wait	启动一个服务器并立刻返回
once	启动服务器并检查一遍所有守护进程
status	显示服务器进程的状态

工作原理

swift-init 工具是一个单独的工具，用来管理任何运行的 OpenStack 对象存储守护进程，便于管理集群。

更多参考

Swift 各种特性和服务的详细介绍，请查看 http://docs.openstack.org/developer/swift/admin_guide.html。

7.3 检查集群健康状况

通过 swift-dispersion-report 工具，可以检查集群的健康状况。这是通过检查分布式的容器集合，以保证对象在集群中的位置正确。

准备工作

确认已登录到 swift-proxy 节点。如果使用 Vagrant 创建了该节点，执行以下命令。


```
vagrant ssh swift-proxy
```

操作步骤

执行以下步骤设置 swift-dispersion 工具报告集群健康状况。

1. 首先，创建 swift-dispersion 工具所需的配置文件/etc/swift/dispersion.conf。

```
[dispersion]
auth_url = https://192.168.100.200:5000/v2.0
auth_user = cookbook:admin
auth_key = openstack
auth_version = 2.0
keystone_api_insecure = yes
```

 当前环境下，我们使用的是 keystone-api-insecure，因为 keystone 端点使用了自签名的证书。这样设置会跳过证书验证环节。

2. 然后，需要创建整个集群中的容器和对象，通过使用 `swift-dispersion-populate` 工具使它们分布在不同的地方。

```
sudo swift-dispersion-populate
```

输出如图 7-1 所示。

```
Created 2621 containers for dispersion reporting, 1m, 0 retries
Created 2621 objects for dispersion reporting, 2m, 0 retries
```

图 7-1

3. 一旦容器和对象设置完毕，就可以运行 `swift-dispersion-report` 命令：

```
sudo swift-dispersion-report
```

产生图 7-2 所示的输出。

```
Querying containers: 1101 of 2622, 37s left, 0 retries

Queried 2622 containers for dispersion reporting, 1m, 0 retries
100.00% of container copies found (7866 of 7866)
Sample represents 1.00% of the container partition space
Queried 2621 objects for dispersion reporting, 30s, 0 retries
There were 2621 partitions missing 0 copy.
100.00% of object copies found (7863 of 7863)
Sample represents 1.00% of the object partition space
```

图 7-2

4. 可以设置一个 cron 作业，重复检测这些容器和对象的健康状况。

```
echo "/usr/bin/swift-dispersion-report" \
| sudo tee -a /etc/cron.hourly/ swift-dispersion-report
```

工作原理

可以通过检测副本是否正确来检查对象的健康。如果 OpenStack 对象存储集群复制一个对象 3 次，其中两次是在正确的位置，那么该对象的健康度为 66.66%。

为保证在集群里有足够的副本，可使用 `swift-dispersion-populate` 工具，它会创建 2621 个容器和对象来填充，因而增加集群大小。一旦到位，就可以设置一个 `corn` 作业，每小时运行一次，以确保集群是一致的，因此，可以很好地体现集群是否健康。

通过设置代理节点（可以访问所有节点）上的 `cron` 作业，我们可以持续检查整个集群的健康状况。在本例中，`cron` 作业每小时运行一次，执行 `swift-dispersion-report` 工具。

7.4 管理 Swift 集群容量

一个地区（zone）是一组与其他节点隔离开来的独立的节点（单独的服务器、网络、电源，甚至地理位置）。Swift 环（ring）可以让 Swift 服务准确定位每个对象，并确保每个副本保存在单独的地区。为了增加存储的容量，可以添加数据备份的额外地区。在本例中，将添加一个存储节点 IP 为 172.16.0.212，第二块磁盘为 `/dev/sdb`，通过 IP 172.16.0.212 用作另一块 OpenStack 对象存储。这个节点组成了这个地区唯一的节点。

要增加额外的容量给现有的地区，要在集群中的每个地区重复进行操作。例如，下面的步骤假定地区 5（z5）不存在，因此在建立环时被创建。为了方便地添加额外的容量给现有区域，在现有的地区（地区 1~4）中指定新服务器。整个指令保持不变。

准备工作

确认已登录到 `swift-proxy` 节点。如果使用 `Vagrant` 创建了该节点，执行以下命令。

```
vagrant ssh swift-proxy
```

操作步骤

为了添加额外的地区到 OpenStack 对象存储集群，执行以下操作。

1. 创建代理服务器。
2. 创建存储节点。

代理服务器

为了创建代理服务器，执行以下操作。

1. 首先，需要将以下条目添加到环中，其中 `STORAGE_LOCAL_NET_IP` 是新节点的 IP

地址, ZONE 是新地区。



确保以下命令在/etc/swfit 目录下执行。

```
cd /etc/swift
```

```
ZONE=5
```

```
STORAGE_LOCAL_NET_IP=172.16.0.212
```

```
WEIGHT=100
```

```
DEVICE=sdb1
```

```
swift-ring-builder account.builder add z$ZONE- $STORAGE_LOCAL_NET_IP:6002/  
$DEVICE $WEIGHT
```

```
swift-ring-builder container.builder add z$ZONE- $STORAGE_LOCAL_NET_IP:6001/  
$DEVICE $WEIGHT
```

```
swift-ring-builder object.builder add z$ZONE- $STORAGE_LOCAL_NET_IP:6000/  
$DEVICE $WEIGHT
```

2. 需要验证环的内容, 执行以下命令。

```
swift-ring-builder account.builder
```

```
swift-ring-builder container.builder
```

```
swift-ring-builder object.builder
```

3. 最后, 重新调整这些环, 这将会耗费一些时间。

```
swift-ring-builder account.builder rebalance
```

```
swift-ring-builder container.builder rebalance
```

```
swift-ring-builder object.builder rebalance
```

4. 执行完毕后, 需要复制 account.ring.gz、container.ring.gz 和 object.ring.gz 到新的存储节点和其他的存储节点。

```
scp *.ring.gz $STORAGE_LOCAL_NET_IP:/tmp  
# And other scp to other storage nodes
```

存储节点

为了创建存储节点, 执行以下操作。

1. 首先复制 account.ring.gz、container.ring.gz 和 object.ring.gz 到/etc/swfit 目录, 确保它们属于 swfit 用户:

```
mv /tmp/*.ring.gz /etc/swift
```

```
chown swift:swift /etc/swift/*.ring.gz
```

如 5.5 节所描述的那样, 在该节点准备存储。

2. 编辑/etc/swift/swift.conf 文件, 使得[swift-hash]部分跟其他所有节点的对应部分匹配。

```
[swift-hash]
# Random unique string used on all nodes
Swift_hash_path_prefix=a4rUmUIgJYXpKhbh
swift_hash_path_suffix=NESuuUEqc6OXwy6X
```

3. 现在需要创建相应的/etc/rsyncd.conf 文件, 添加以下内容。

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 172.16.0.212

[account]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/account.lock

[container]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/container.lock

[object]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/object.lock
```

4. 执行以下命令, 启用和启动 rsync。

```
sed -i 's/=false/=true/' /etc/default/rsync
service rsync start
```

5. 需要创建/etc/swift/account-server.conf 文件, 并为其添加以下内容。

```
[DEFAULT]
bind_ip = 172.16.0.212
workers = 2

[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account

[account-replicator]
```



```
[account-auditor]
```

```
[account-reaper]
```

6. 创建/etc/swift/container-server.conf 文件，并为其添加以下内容。

```
[DEFAULT]
bind_ip = 172.16.0.212
workers = 2
```

```
[pipeline:main]
pipeline = container-server
```

```
[app:container-server]
use = egg:swift#container
```

```
[container-replicator]
```

```
[container-updater]
```

```
[container-auditor]
```

7. 最后，创建/etc/swift/object-server.conf 文件，并为其添加以下内容。

```
[DEFAULT]
bind_ip = 172.16.0.212
workers = 2
```

```
[pipeline:main]
pipeline = object-server
```

```
[app:object-server]
use = egg:swift#object
```

```
[object-replicator]
```

```
[object-updater]
```

```
[object-auditor]
```

8. 现在可以启动这个存储节点，这个节点已被配置成为我们的第5个地区。

```
swift-init all start
```

工作原理

通过添加额外的节点或地区进行扩容，需要完成以下两步。

1. 在代理服务器上配置地区和节点。
2. 配置存储节点。

对于每个存储节点和这些存储节点上的设备，运行下面的命令，为新的地区增添存储

节点和设备：

```
swift-ring-builder account.builder add zzone-storage_ip:6002/device weight
swift-ring-builder container.builder add zzone-storage_ip:6001/device weight
swift-ring-builder object.builder add zzone-storage_ip:6000/device weight
```

一旦在代理节点上完成这些配置，就需要重新平衡环。这样会更新对象环、账户环和容器环。复制更新后的 gzip 压缩文件，以及环境中用到的 swift 散列值到存储节点。

在存储节点上，需执行以下步骤。

1. 配置磁盘（分区和格式化 XFS）。
2. 配置并启动 rsyncd。
3. 配置账户、容器和对象服务。
4. 启动存储节点上的 OpenStack 对象存储服务。

OpenStack 对象存储环境内的数据会重新分配到这个新地区的节点。

7.5 从集群中删除节点

有时我们需要减少容量，或者将一个失败的节点从服务中移除。可以通过从集群中的地区中移除节点来实现。接下来，将移除 z5 中的 172.16.0.212 节点，也就是存储设备 /dev/sdb1。

准备工作

确认已登录到 swift-proxy 节点。如果使用 Vagrant 创建了该节点，执行以下命令。

```
vagrant ssh swift-proxy
```

操作步骤

执行以下操作，从一个地区中删除一个存储节点，需要在代理服务器配置中进行更改。

代理服务器

这里需要对代理服务器的配置做调整，执行以下操作。

1. 要从 OpenStack 对象存储删除一个节点，先设定其 weight 为 0，重新调整环时，数据会从这个节点搬走。

```
cd /etc/swift
```

```
swift-ring-builder account.builder set_weight z5-172.16.0.212:6002/sdb1 0
swift-ring-builder container.builder set_weight z5-172.16.0.212:6001/sdb1 0
swift-ring-builder object.builder set_weight z5-172.16.0.212:6000/sdb1 0
```

2. 重新调整环。

```
swift-ring-builder account.builder rebalance
swift-ring-builder container.builder rebalance
swift-ring-builder object.builder rebalance
```

3. 完成之后，可以从该环中把节点从地区中移除。

```
swift-ring-builder account.builder remove z5-172.16.0.212:6002/sdb1
swift-ring-builder container.builder remove z5-172.16.0.212:6001/sdb1
swift-ring-builder object.builder remove z5-172.16.0.212:6000/sdb1
```

4. 然后复制 `account.ring.gz`、`container.ring.gz` 和 `object.ring.gz` 文件到集群中的其余节点。现在可以停用此存储节点，移除这个物理设备。

工作原理

从 OpenStack 对象存储集群手动移除一个节点需要以下 3 步。

1. 使用 `swift-ring-builder <ring> set_weight` 命令设置节点权重为 0，所以数据不会再复制过去。

2. 重新调整环，更新数据。

3. 使用 `swift-ring-builder <ring> remove` 命令从 OpenStack 对象存储集群上移除节点。完成之后，就可以删除该节点。重复以上步骤删除该地区的每一个节点。

7.6 检测和更换故障硬盘

如果不能访问存储数据的硬盘驱动器，OpenStack 对象存储也不会有多少用处。所以必须要能够检测和更换故障硬盘。OpenStack 对象存储可以通过 `swift-drive-audit` 命令检测硬件故障。这有助于检测故障，及时替换故障硬盘，极大地提高了系统的健康度和性能。

准备工作

确认已登录到 `swift-proxy` 节点。如果使用 Vagrant 创建了该节点，执行以下命令。

```
vagrant ssh swift-proxy
vagrant ssh swift-01
```


操作步骤

要检测出故障的硬盘驱动器，需要执行以下操作。

存储节点

我们需要执行以下操作，在存储节点上做些修改。

1. 首先，需要配置一个 cron 作业，监控 /var/log/kern.log 文件来检测存储节点中的硬盘错误。为此，需要创建一个配置文件 /etc/swift/swift-drive-audit.conf。

```
[drive-audit]
log_facility=LOG_LOCAL0
log_level=INFO
device_dir=/srv/node
minutes=60
error_limit=1
```

2. 然后，添加一个 cron 作业，每小时执行一次 swift-drive-audit，或根据环境需要设置。

```
echo '/usr/bin/swift-drive-audit /etc/swift/swift-drive-audit.conf' | sudo tee -a /etc/cron.hourly/swift-drive-audit
```

3. 当一个磁盘被检测出问题，脚本会卸载该磁盘，OpenStack 对象存储会绕过这个问题。因此，当一个磁盘被标记出问题并脱机时，就可以替换它。



没有 swift-drive-audit 自动化监控的话，必须要手动执行该操作，以确保硬盘被卸载并从环中移除。

4. 一旦硬盘被物理替换，就可以遵循 7.4 节中描述的指令，将节点和硬盘添加回集群中。

工作原理

可以设置一个 cron 作业，每小时执行一次 swift-drive-audit 来自动检测失败的硬盘驱动器。这个作业会检查 /var/log/kern.log 文件，检测磁盘故障。Ubuntu 14.04 默认会记录硬件系统信息至该文件。有了这个脚本，就可以检测磁盘故障、卸载驱动器不再使用它、更新环，让数据不再被复制和存储到该驱动器上。

一旦驱动器从环中被删除后，就可以在对该设备进行维护并替换该驱动器。

新驱动器到位后，就可以通过添加回环中把设备重新放回存储节点上的服务中。然后通过 swift-ring-builder 命令重新调整环。

7.7 收集使用情况统计数据

通过添加 swift-recon 中间件到 object-server 配置中, OpenStack 对象存储可以报告使用情况。通过使用 swift-recon 工具, 可以查询这些指标。

准备工作

确保登录到所有节点, 并安装了相应的软件包, 配置好了 Swift 服务。如果使用 Vagrant 创建了该环境, 可以执行如下命令访问所有节点。

```
vagrant ssh swift-proxy
vagrant ssh swift-01
vagrant ssh swift-02
vagrant ssh swift-03
vagrant ssh swift-04
vagrant ssh swift-05
```

操作步骤

要收集 OpenStack 对象存储集群的使用数据, 需要执行以下步骤。

1. 首先, 需要修改存储节点上的 /etc/swift/object-server.conf 配置文件, 添加 swift-recon 中间件, 如下所示。

```
[DEFAULT]
bind_ip = 0.0.0.0
workers = 2

[pipeline:main]
pipeline = recon object-server

[app:object-server]
use = egg:swift#object

[object-replicator]

[object-updater]

[object-auditor]

[filter:recon]
use = egg:swift#recon
recon_cache_path = /var/cache/swift
```



在示例 Vagrant 环境中, 我们在一个主机上运行了多个 Swift 节点。因此, 需要在每个节点上的 /etc/swift/object_server 目录下编辑节点对象配置文件。

2. 然后, 使用 `swfit-init` 重启 `object-server` 服务。

```
swift-init object-server restart
```

运行该命令的同时, 可以在代理服务器上使用 `swfit-recon` 工具获取使用统计数据。

□ 硬盘使用:

```
swift-recon -d
```

该命令会报告集群中硬盘的使用情况。

```
swift-recon -d -z5
```

该命令会报告地区 5 上硬盘的使用情况。

□ 平均负载:

```
swift-recon -l
```

该命令会报告集群中的平均负载情况。

```
swift-recon -l -z5
```

该命令会报告地区 5 上节点的平均负载情况。

□ 隔离统计:

```
swift-recon -q
```

这会报告集群中任意一个隔离的容器、对象和账户信息。

```
swift-recon -q -z5
```

这将只报告地区 5 里的相关信息。

□ 检测卸载设备:

```
swift-recon -u
```

这将检查集群中任意没有卸载的驱动器。

```
swift-recon -z5 -u
```

这将只报告地区 5 里的相关信息。

□ 检测复制指标:

```
swift-recon -r
```

这将只报告集群内的复制状态。


```
swift-recon -r -z5
```

这将只报告地区 5 里节点的相关信息。

可以用一个命令执行所有这些操作得到集群的所有遥测数据。

```
swift-recon --all
```

在后面添加一个-z5，就可以只获得地区 5 里的节点信息。

```
swift-recon --all -z5
```

最后，可以定期运行一个 cron 作业，检查环境中异步待处理的对象信息。可以这样创建 cron 作业：

```
*/5 * * * * swift /usr/bin/swift-recon-cron /etc/swift/objectserver.conf
```

工作原理

为了统计 OpenStack 对象存储使用信息，添加一个 swift-recon 中间件收集度量信息。将以下几行添加到每个存储节点的/etc/swift/object-server.conf 文件中，就可以将中间件添加到对象服务器。

```
[pipeline:main]
pipeline = recon object-server

[filter:recon]
use = egg:swift#recon
recon_cache_path = /var/cache/swift
```

配置好后重启对象服务器，就可以通过 swift-recon 工具查询此遥测数据。可以把集群作为一个整体收集数据，或使用-z 选项指定某个特定的地区，收集统计数据。

需要注意的是，还可以通过在命令行中指定-all 标志或追加多个参数来收集所有或多种统计数据。例如，要从地区 5 的节点中收集平均负载和复制数据，可以执行以下命令。

```
swift-recon -r -l -z5
```

第 8 章

Cinder——OpenStack 块存储

本章将讲述以下内容：

- ☐ 配置 Cinder 卷服务
- ☐ 为 Cinder 卷配置 OpenStack 计算服务
- ☐ 创建卷
- ☐ 为实例添加卷
- ☐ 从实例中分离卷
- ☐ 删除卷
- ☐ 配置第三方卷服务
- ☐ 使用 Cinder 快照
- ☐ 从卷启动

8.1 简介

写到磁盘上运行中的虚拟机实例的数据是非持久的，也就是说，如果终止正在运行的虚拟机，磁盘上的数据就会丢失。卷是可以添加到 OpenStack 计算实例的持久存储；最好的类比是，可以将卷看作给每一个虚拟机实例挂载一个 USB 设备。与 USB 设备类似，卷每次只允许被挂载到一台计算机上使用的。

在之前的 OpenStack 版本中，卷服务是通过 `cinder-volume` 提供的，它逐渐演进为现在的 OpenStack 块存储服务（OpenStack Block Storage），即 Cinder 项目。OpenStack 块

存储与亚马逊 EC2 中的 EBS (Elastic Block Storage) 非常相似, 它们之间的区别在于存储卷暴露给虚拟机实例的方式。在 OpenStack 计算服务中, 可以通过 iSCSI 暴露的 LVM 卷组 cinder-volumes 对卷进行管理。因此, 每个运行 Cinder 卷服务的主机必须拥有该卷组。

Cinder 卷是运行的服务的名称, cinder-volumes 是被 Cinder 卷服务使用的 LVM 卷组的名称, 有时会导致 OpenStack 块存储管理产生混淆。

在本章中, 我们将再向 OpenStack 块存储服务添加一个节点。图 8-1 说明了最新的环境以及 Cinder 所处的位置。

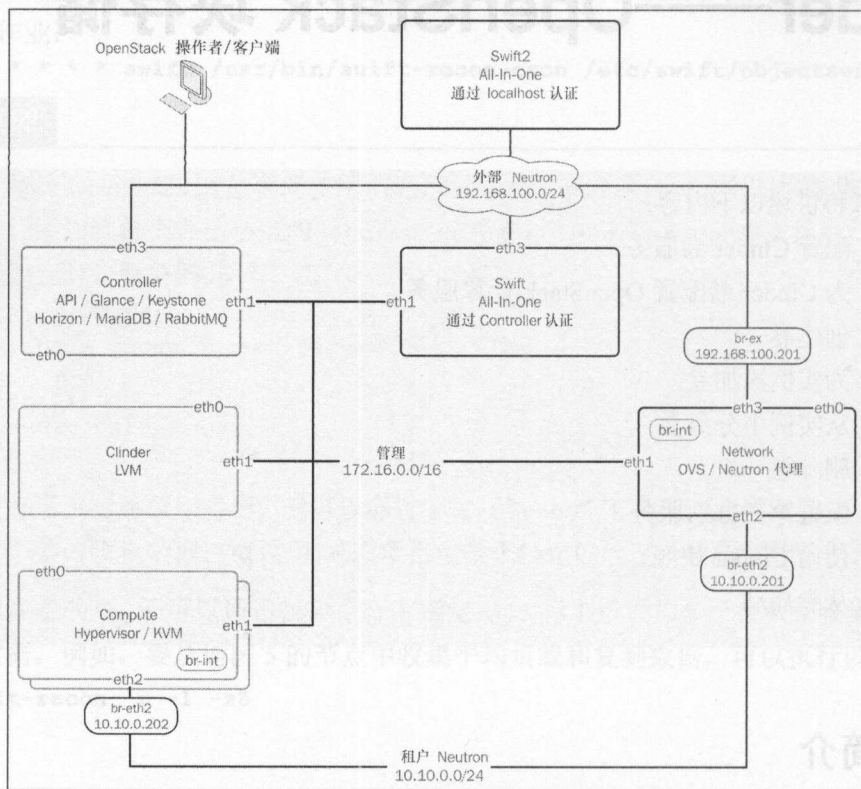


图 8-1

8.2 配置 Cinder 卷服务

本节会配置一台运行 Ubuntu 14.04 LTS 的新服务器用于托管卷, 并学习将卷挂载到实例时所需满足的依赖服务。

准备工作

为了使用 Cinder 卷，需要拥有一台运行 Ubuntu 14.04 LTS 的主机。这台主机将提供回环文件系统（loopback filesystem），在后者的基础上打造 LVM 卷，并安装 Cinder 所需的服务。

如果你使用本书配套的 Vagrant 环境，请执行如下命令登录到 cinder 节点。

```
vagrant up cinder
vagrant ssh cinder
```

本节中，OpenStack 块存储和 Cinder 可互换使用。

操作步骤

首先，需要配置好一个回环文件和 LVM。之后，将安装并配置依赖服务，如 open-iscsi。最后，配置 Cinder。

要为 Cinder 卷添加一个新的主机，需要执行以下步骤。

1. 首先登录到新主机。

2. 使用如下命令安装依赖。

```
# Install some dependencies
sudo apt-get update
sudo apt-get install linux-headers-`uname -r` \
    build-essential python-mysqldb xfsprogs

sudo apt-get install cinder-api cinder-scheduler \
    cinder-volume open-iscsi python-cinderclient tgt \
    iscsitarget iscsitarget-dkms
```

3. 现在需要使用如下命令重启 open-iscsi。

```
sudo service open-iscsi restart
```

要创建回环文件系统并为使用 cinder-volume 设置 LVM，执行以下步骤。

4. 首先，创建一个 5 GB 的文件，用来做回环文件系统。

```
dd if=/dev/zero of=cinder-volumes bs=1 count=0 seek=5G
```

然后，创建回环文件系统。

```
sudo losetup /dev/loop2 Cinder-volumes
```

5. 最后，为 cinder-volume 创建 LVM。

```
sudo pvcreate /dev/loop2
```

```
sudo vgcreate cinder-volumes /dev/loop2
```

重点提醒，这并不是一个持久文件系统。这里仅做演示使用，在生产环境中，应该使用实际的卷，而不是回环文件，并持久挂载。

工作原理

为了使用 cinder-volume，需要准备一个设置为 LVM 的磁盘或者分区，取名为 cinder-volumes。在本书中，通过增加一个设置为 LVM 卷的回环文件系统来简单模拟。在实际环境中，步骤相同。此处只是简单地将一个新增分区设置为 8e (Linux LVM) 格式，然后把该分区添加到命名为 cinder-volumes 的卷组中。

完成上述的步骤之后，接下来安装 cinder-volume 工具以及支撑服务。cinder-volume 使用 iSCSI 做为软件模拟磁盘阵列管理，实现将卷分配给虚拟机实例，所以还需要安装 iSCSI 的所有依赖环境以及 iSCSI。

8.3 为 Cinder 卷配置 OpenStack 计算服务

需要让 OpenStack 计算服务知道新增了 Cinder 卷服务。

准备工作

由于要在多节点环境中进行设置，所以需要登录到控制节点、计算节点和 Cinder 节点上。

如果使用的是本书配套的 Vagrant 环境，可执行如下命令登录到这些节点。

```
vagrant ssh controller
vagrant ssh cinder
```

本节需要创建一个 openrc 文件。每个节点都需要该文件。为此，打开文本文件 openrc，添加以下内容。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_KEY=/path/to/akey.pem
export OS_CACERT=/path/to/ca.pem
```

操作步骤

在多节点环境中，需要配控制节点、计算节点和 Cinder 节点。因此，按顺序执行以下命令。

要为 cinder-volume 配置 OpenStack 控制节点，需要执行以下步骤。

1. 在多节点配置中，OpenStack 控制节点负责认证（keystone）和部署 Cinder 数据库。首先，登录到控制节点，配置认证。

```
source openrc
keystone service-create \
    --name volume \
    --type volume \
    --description 'Volume Service'

CINDER_SERVICE_ID=$(keystone service-list | awk '/\ volume\
/ {print $2}')
```

```
PUB_CINDER_ENDPOINT="192.168.0.211"
INT_CINDER_ENDPOINT="172.16.0.211"

PUBLIC="http://$PUB_CINDER_ENDPOINT:8776/v1/$(tenant_id)s"
ADMIN="http://$INT_CINDER_ENDPOINT:8776/v1/$(tenant_id)s"
INTERNAL=$PUBLIC

keystone endpoint-create \
    --region RegionOne \
    --service_id $CINDER_SERVICE_ID \
    --publicurl $PUBLIC \
    --adminurl $ADMIN \
    --internalurl $INTERNAL

keystone user-create \
    --name cinder \
    --pass cinder \
    --tenant_id $SERVICE_TENANT_ID \
    --email cinder@localhost --enabled true

CINDER_USER_ID=$(keystone user-list \
    | awk '/\ cinder \ / {print $2}')
```

```
keystone user-role-add \
    --user $CINDER_USER_ID \
    --role $ADMIN_ROLE_ID \
    --tenant_id $SERVICE_TENANT_ID
```

2. 接下来，为 Cinder 创建 MariaDB/MySQL 数据库。

```
MYSQL_ROOT_PASS=openstack
MYSQL_CINDER_PASS=openstack

mysql -uroot -p$MYSQL_ROOT_PASS \
    -e 'CREATE DATABASE cinder;'

mysql -uroot -p$MYSQL_ROOT_PASS \
    -e "GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%';"

mysql -uroot -p$MYSQL_ROOT_PASS \
    -e "SET PASSWORD FOR 'cinder'@'%'=
PASSWORD('$MYSQL_CINDER_PASS');"

```


3. 编辑/etc/nova/nova.conf 文件, 在[Default]部分加入如下内容。

```
volume_driver=nova.volume.driver.ISCSIDriver

enabled_apis=ec2,osapi_compute,metadata
volume_api_class=nova.volume.cinder.API
iscsi_helper=tgtadm
```

4. 重启 nova 服务。

```
for P in $(ls /etc/init/nova* | cut -d '/' -f4 | cut -d '.' -f1)
do
    sudo stop ${P}
    sudo start ${P}
done
```

要为 Cinder 配置 OpenStack 计算节点, 需要执行以下步骤。

1. 接下来要配置的是 OpenStack 计算节点。我们将介绍如何配置第一个节点, 然后在所有计算节点上复制该配置。首先登录到一个计算节点。

```
vagrant ssh compute-01
```

2. 编辑/etc/nova/nova.conf, 添加以下内容到[Default]部分下。

```
volume_driver=nova.volume.driver.ISCSIDriver
enabled_apis=ec2,osapi_compute,metadata
volume_api_class=nova.volume.cinder.API
iscsi_helper=tgtadm
```

3. 重启 nova 服务。

```
for P in $(ls /etc/init/nova* | cut -d '/' -f4 | cut -d '.' -f1)
do
    sudo stop ${P}
    sudo start ${P}
done
```

要配置 Cinder 节点来使用 cinder-volume, 需要登录到 Cinder 节点并执行以下步骤。

1. 编辑/etc/cinder/cinder.conf 文件, 加入以下内容, 在内部地址上启用与 Keystone 之间的通信。

```
[keystone_auth_token]
auth_uri = https://192.168.100.200:35357/v2.0/
identity_uri = https://192.168.100.200:5000
admin_tenant_name = service
admin_user = cinder
admin_password = cinder
insecure = True
```

2. 接下来, 修改/etc/cinder/cinder.conf 来配置数据库、iSCSI 和 RabbitMQ。确保 cinder.conf 文件中有以下内容。

```
[DEFAULT]
rootwrap_config=/etc/cinder/rootwrap.conf
```

```
[database]
backend=sqlalchemy
connection = mysql://cinder:openstack@172.16.0.200/cinder
```

```
iscsi_helper=tgtadm
volume_name_template = volume-%s
volume_group = cinder-volumes
verbose = True
auth_strategy = keystone
```

Add these when not using the defaults.

```
rabbit_host = 172.16.0.200
```

```
rabbit_port = 5672
```

```
state_path = /var/lib/cinder/
```

3. 最后，同步 Cinder 数据库并重启 Cinder 服务。

```
cinder-manage db sync
```

```
cd /etc/init.d/; for i in $( ls cinder-* ); do sudo service $i
restart; done
```

工作原理

在多节点环境中，需要在各个节点配置，才能让 OpenStack 使用 Cinder 卷服务。在 OpenStack 控制节点中，创建了一个 Keystone 服务、端点（endpoint）和用户。另外，分配了一个 cinder 用户，在服务租户中拥有管理员角色。在控制节点中，创建了一个 cinder MySQL 数据库并修改 nova.conf 赋予 cinder 用户相应的权限。

在计算节点，修改相对简单，只需要修改 nova.conf 修改 Cinder 权限。

最后，配置 Cinder 节点。方法是启动 Keystone 和初始化 Cinder 数据库，连接 Cinder 服务到 MySQL 数据库。之后重启 Cinder 服务。

8.4 创建卷

现在拥有一个可用的 Cinder 卷服务，也就可以给虚拟机实例创建和分配存储卷了。实验环境中使用的是在 Ubuntu 下的 Cinder 客户端工具 python-cinderclient，给租户（项目）创建存储卷。

准备工作

确保已登录到可以使用 Cinder 客户端工具的 Ubuntu 客户机上。如果使用的是本书配套的 Vagrant 环境，可在 cinder 节点上使用这些工具。

vagrant ssh cinder

本节需要创建一个 openrc 文件。每个节点都需要该文件。为此,打开文本文件 openrc,添加以下内容。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_KEY=/path/to/akey.pem
export OS_CACERT=/path/to/ca.pem
```

如果没有安装工具,则可以通过如下命令安装。

```
sudo apt-get update
sudo apt-get install python-cinderclient
```

操作步骤

要使用 Cinder 客户端创建卷,需要执行如下步骤。

1. 首先,给虚拟机实例创建一个卷。

```
source openrc
cinder create --display-name cookbook 1
```

2. 执行完之后,命令行返回以下输出,如图 8-2 所示。

Property	Value
attachments	[]
availability_zone	nova
bootable	false
created_at	2015-07-03T20:55:40.704025
display_description	None
display_name	cookbook
encrypted	False
id	06ab21fb-bc76-4216-9491-6c1918c1dab2
metadata	{}
size	1
snapshot_id	None
source_volid	None
status	creating
volume_type	None

图 8-2

工作原理

为项目 cookbook 创建 Cinder 卷非常简单,使用 Cinder 客户端,带有 create 选项,语法如下:

```
cinder create --display_name volume_name size_Gb
```

这里的 volume_name 可以是任意不带空格的名称。可以通过 LVM 工具查看逻辑卷

cinder-volumes 详情。

```
sudo lvdisplay cinder-volumes
```

输出如图 8-3 所示。

```

--- Logical volume ---
LV Path                /dev/cinder-volumes/volume-06ab21fb-bc76-4216-9491-6c1918c1dab2
LV Name                 volume-06ab21fb-bc76-4216-9491-6c1918c1dab2
VG Name                 cinder-volumes
LV UUID                 gg1gKl-X34n-q9xU-xGbc-7C8x-tIxF-yaPU6B
LV Write Access         read/write
LV Creation host, time cinder, 2015-07-03 15:55:40 -0500
LV Status                available
# open                  0
LV Size                 1.00 GiB
Current LE              256
Segments                1
Allocation              inherit
Read ahead sectors      auto
  - currently set to    256
Block device            252:2

```

图 8-3

注意，LV 的名称与使用 Cinder 创建的卷的 ID 相同。

8.5 为实例添加卷

通过刚才的配置，已经有了可用的卷，可以让任意的虚拟机使用它。通过 Nova 客户端工具中的 nova volume-attach 选项来使用卷。

准备工作

确保已登录到可以使用 Nova 客户端工具的 Ubuntu 客户机上。如果使用的是本书配套的 Vagrant 环境，可在 Controller 节点上使用这些工具。

```
vagrant ssh controller
```

本节需要创建一个 openrc 文件。在每个需要的节点创建 openrc 文件。为此，打开文本文件 openrc，添加以下内容。

```

export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_KEY=/path/to/akey.pem
export OS_CACERT=/path/to/ca.pem

```

如果没有安装工具，则可以通过如下命令安装。

```

sudo apt-get update
sudo apt-get install python-novaclient

```

操作步骤

要使用 Nova 客户端为实例增加卷，需要执行以下步骤。

1. 如果还没有启动实例，请启动实例。启动之后，运行 `nova list` 命令查看实例 ID。

```
source openrc
nova list --fields name
```

输出如图 8-4 所示。

ID	Name
f9659289-82f1-435f-98fc-add99c7a611b	test1

图 8-4

2. 使用实例 ID，就可以添加卷到运行的实例上。

```
nova volume-attach <instance_id> <volume_id> /dev/vdc
```

3. 执行成功后，会返回卷名。可以登录到运行的虚拟机上看到卷已可用。

```
sudo fdisk -l /dev/vdc
```

4. 在运行的实例中可以看到有 1 GB 的空间可用。下面，可以通过格式化、设置挂载点，就像使用一个新硬盘一样使用它了。

```
sudo mkfs.ext4 /dev/vdc
sudo mkdir /mnt1
sudo mount /dev/vdc /mnt1
```

5. 这时，可以查看到新的卷已挂载到了 `/mnt1` 下。

```
df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/vda	1.4G	602M	733M	46%	/
devtmpfs	248M	12K	248M	1%	/dev
none	50M	216K	50M	1%	/run
none	5.0M	0	5.0M	0%	/run/lock
none	248M	0	248M	0%	/run/shm
/dev/vdb	5.0G	204M	4.6G	5%	/mnt
/dev/vdc	1.0G	204M	784M	5%	/mnt1

工作原理

使用 Cinder 卷其实和在本机使用一个 USB 磁盘没什么太大差别；一个 Cinder 卷一次只能连接到一个实例上，而且必须格式化并挂载才可使用。

在 Nova 客户端工具中，`volume-attach` 选项的语法如下。

```
nova volume-attach instance_id volume_id device
```

`instance_id` 参数是通过 `nova list` 获得的 ID，是卷将连接到的实例。`volume_id` 则是实例中用于挂载卷的设备，可以使用 `nova volume-list` 得到。该设备是在虚拟机中便于挂载使用卷而创建的。

8.6 从实例中分离卷

因为 Cinder 卷一次只可用于一台计算机，将其与这台计算机分离后，才可用于其他的计算机。在分离卷时，会用到另一个 Nova 客户端选项 `volume-detach`。

准备工作

确保已登录到可以使用 Nova 客户端工具的 Ubuntu 客户机上。如果使用的是本书配套的 Vagrant 环境，可在 controller 节点上使用这些工具。

```
vagrant ssh controller
```

本节需要创建一个 `openrc` 文件。每个节点都需要该文件。为此，打开文本文件 `openrc`，添加以下内容。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_KEY=/path/to/akey.pem
export OS_CACERT=/path/to/ca.pem
```

如果没有安装工具，则可以通过如下命令安装。

```
sudo apt-get update
sudo apt-get install python-novaclient
```

操作步骤

要使用 Nova 客户端工具分离卷，需要执行如下步骤。

1. 首先，需要先确认虚拟机使用了哪些卷，执行 `nova volume-list` 命令：

```
nova volume-list
```

该命令返回结果如图 8-5 所示。

ID	Status	Display Name	Size	Volume Type	Attached to
06ab21fb-bc76-4216-9491-6c1918c1dab2	available	cookbook	1	None	

图 8-5

2. 在已经挂载卷的实例上, 必须先执行 `umount` (如使用前面的示例中, 应该是在 `/mnt1`)。

```
sudo umount /mnt1
```

3. 回到已经安装了 Nova 客户端的 Ubuntu 客户机, 现在可以分离该卷与虚拟机的关联了。

```
nova volume-detach <instance_id> <volume_id>
```

此时, 可以将其挂载到其他运行的虚拟机实例上, 卷中数据依然保留。

工作原理

分离 Cinder 卷就好比将 USB 磁盘从计算机上拔出, 先将卷从运行的虚拟机实例上卸载, 然后使用 Nova 客户端命令 `nova volume-detach` 将卷从运行实例上分离出来。

使用 `nova volume-detach` 命令的语法如下。

```
nova volume-detach instance_id volume_id
```

`instance_id` 是通过 `nova volume-list` 返回的 `attached to` 列得到的, 指的是从哪个实例上分离卷。

`volume_id` 参数则可以使用 `nova volume-list` 获取的 ID 列得到。

8.7 删除卷

在某些情况下, 可能不再需要之前创建的卷, 需要将其从系统中完全的清理掉。通过 Nova 客户端工具中的 `volume-delete` 可以很方便地做到这些。

准备工作

确保已登录到已经安装 Nova 客户端的 Ubuntu 主机上, 且在 OpenStack 环境认证信息中授权。如果使用的是本书配套的 Vagrant 环境, 可使用如下命令登录到控制节点。

```
vagrant ssh controller
```



注意, 这是一次性操作。除非已做好备份, 否则数据将永久删除。
确保的确想要删除卷。

操作步骤

要使用 Nova 客户端删除一个卷, 需要执行以下步骤。

1. 首先，需要识别出要删除的卷。

```
cinder list
```

2. 通过上面命令返回的卷 ID，可以很简单地将卷从系统中删掉。

```
cinder delete <volume_id>
```

删除完毕后，该命令会返回被删除卷的详细信息。

工作原理

删除卷后，系统中将无法再使用原来的 LVM 卷。删除 LVM 卷只需要把要删除的卷名作为参数传给 `nova volume-delete` 命令（如果是用 Nova 客户端工具），即可完成删除卷的操作。当然，首先要确认被删除的卷没被使用。

8.8 配置第三方卷服务

OpenStack 块存储项目 Cinder，默认依赖 Linux 的 iSCSI。虽然该项目很健壮，但是有时需要将 OpenStack 集成到现有环境，或希望使用第三方存储服务提供的更强大特性。本节将介绍如何配置 Cinder 使用其他存储提供商插件。

准备工作

确保已登录到已经安装 `cinder-api` 服务的 Ubuntu 主机上，且在 OpenStack 环境认证信息中授权。



本例将说明在 Cinder 中使用 NFS 后端。值得指出的是，尽管配置简单直接，但在使用其他第三方服务时，应参考服务商的文档。

操作步骤

要将 Cinder 卷的驱动更改为 NFS，需要执行如下步骤。

1. 需要一台合理配置的 NFS 服务器，并在运行有 `cinder-api` 服务的节点上创建一个名为 `/etc/cinder/nfsshare` 的文件，针对每个 NFS 卷，都要有一行如下格式的配置行。

```
cinder.book:/exports
```

2. 下一步，需要编辑 `/etc/cinder/cinder.conf` 文件，加入如下行。

```
volume_driver = cinder.volume.drivers.nfs.NfsDriver
```

该文件还应包含如下行。

```
nfs_shares_config = /etc/cinder/nfsshare
```

最终的文件内容如下所示。

```
[DEFAULT]
rootwrap_config=/etc/cinder/rootwrap.conf
api_paste_config = /etc/cinder/api-paste.ini
volume_driver = cinder.volume.drivers.nfs.NfsDriver
nfs_shares_config = /etc/cinder/nfsshare

verbose = True
use_syslog = True
syslog_log_facility = LOG_LOCAL0

auth_strategy = keystone

rabbit_host = 172.16.0.200
rabbit_port = 5672
state_path = /var/lib/cinder/

[database]
backend=sqlalchemy
connection = mysql://cinder:openstack@172.16.0.200/cinder

[keystone_authtoken]
auth_uri = https://192.168.100.200:35357/v2.0/
identity_uri = https://192.168.100.200:5000
admin_tenant_name = service
admin_user = cinder
admin_password = cinder
signing_dir = \${state_path}/keystone-signing
insecure = True
```

3. 最后，重启 Cinder 服务。

```
cd /etc/init/; for c in $( ls cinder-* | cut -d '.' -f1 ) ; \
do sudo stop $c; start $c; done
```

工作原理

/etc/cinder/nfsshare 文件会告知 Cinder 在分配卷时连接到哪个 NFS 服务器，而 volume_driver 则告诉 Cinder 要使用其他的存储后端。nfs_shares_config 变量提供 Cinder 服务所需的其他配置细节。服务重启之后，Cinder 就能够使用指定的 NFS 服务器用于卷存储。

8.9 使用 Cinder 快照

在 Cinder 中，卷快照可以非破坏式地对卷进行复制，支持卷备份。还使得更复杂的备份特性成为可能，提供了从指定快照启动实例的能力。

本节将介绍如何创建、更新和删除快照。

准备工作

确保已登录到安装了 Cinder 命令行工具的 Ubuntu 主机上,并且在 OpenStack 环境认证信息中授权。

操作步骤


1. 创建快照时,卷不能连接到实例。可使用 `cinder list` 命令,查看当前卷:

```
$ cinder list
```

输出如图 8-6 所示。

ID	Status	Display Name	Size	Volume Type	Bootable	Attached to
b3e0f6b2-19cb-436f-a190-4b5c66ba2daf	available	demo	1	None	false	

图 8-6

 如果要做快照的卷的状态为 in-use,则需要使用 8.6 节所列出的操作分离卷。

2. 由于当前卷的状态为可用,接着使用 `cinder snapshot-create` 命令创建卷的快照:

```
cinder snapshot-create b3e0f6b2-19cb-436f-a190-4b5c66ba2daf
```

输出如图 8-7 所示。

Property	Value
created_at	2015-03-03T03:34:28.863739
display_description	None
display_name	None
id	855de1d1-1e43-4b19-93eb-5c60059890a7
metadata	{}
size	1
status	creating
volume_id	b3e0f6b2-19cb-436f-a190-4b5c66ba2daf

图 8-7

3. 快照创建完成后,可将其重新连接到实例,继续操作。如果快照是持续测试/验证过程或备份方案的一个环节,则需要使用新数据更新快照。为此,使用 `cinder snapshot-reset-state` 命令,如果运行成功则不会产生输出。

4. 最后,需要删除快照。可使用 `cinder snapshot-delete` 命令。

```
cinder snapshot-delete 63c3173b-4f30-4240-99f2-fd2e82cb757e
```

通过下面的命令，确认卷是否成功删除：

```
cinder snapshot-list
```

输出如图 8-8 所示。

ID	Volume ID	Status	Display Name	Size

图 8-8

工作原理

Cinder 卷快照提供了灵活克隆卷进行备份或连接至其他实例的方式。本节用到的 cinder snapshot-系列命令（具体包括 cinder snapshot-create, cinder snapshot-list, cinder snapshot-reset-state 和 cinder snapshot-delete）会指示 Cinder 使用存储驱动器执行以下快照操作：创建、列举、更新和删除。快照的具体实现根据底层驱动而定。

8.10 从卷启动

对 OpenStack 运营而言，从卷启动有多个好处。这样可以为实例提供一层弹性，还可以在无法使用 Libvirt 做磁盘迁移时启用实例的热迁移。

准备工作

确保已登录到安装了 Cinder 命令行工具的 Ubuntu 主机上，并且在 OpenStack 环境认证信息中授权。

操作步骤

要从卷启动实例，首先需要选择从哪个镜像启动，以及选择实例的类型。具体步骤如下：

1. 获得启动所需的镜像 UUID：

```
nova image-list
```

命令的输出如图 8-9 所示。

ID	Name	Status	Server
bbcf92a3-658b-4185-8686-7fb2f77b66a0	cirros-image	ACTIVE	
f5eba890-4660-4ee8-92d0-d7972510b7d6	trusty-image	ACTIVE	

图 8-9

2. 获得实例类型 m1.tiny 的 ID:

nova flavor-list

命令的输出如图 8-10 所示。

ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor	Is_Public
1	m1.tiny	512	1	0		1	1.0	True
2	m1.small	2048	20	0		1	1.0	True
3	m1.medium	4096	40	0		2	1.0	True
4	m1.large	8192	80	0		4	1.0	True
5	m1.xlarge	16384	160	0		8	1.0	True

图 8-10

3. 由于试验环境配置了两个网络，需要选择实例连接到哪一个。首先，列出可用的网络：

neutron net-list

这将列出可用的网络，如图 8-11 所示。

id	name	subnets
0375e772-b021-425c-bc17-5a3263247fb8	cookbook_network_1	37ca3149-ee6b-4288-b074-03a4e1635b7c 10.200.0.0/24
706c9118-68d9-4751-932c-3dc94ec6f4ed	ext_net	889c1ef7-09af-48d2-85bf-6971446e2eb7 192.168.100.0/24

图 8-11

4. 最后，执行 nova boot 命令。

```
nova boot \
  --flavor m1.tiny \
  --block-device source=image,id=trusty-
image,shutdown=preserve,dest=volume,size=15,bootindex=0 \
  --key_name demokey \
  --nic net-id=0375e772-b021-425c-bc17-5a3263247fb8 \
  --config-drive=true \
  Cookbook_Instance
```

工作原理

在最后一步中，向 nova boot 命令传递了许多新参数，让 Nova 在启动镜像时使用 Cinder。具体来说，--block-device 以及 source、id、shutdown、destination、size 和 boot index 等子参数告诉 Nova 从哪一个镜像（source 和 id）启动，在关闭实例时保存 Cinder 卷（shut down），目标卷所处的位置和大小（dest 和 size）。

第 9 章

深入 OpenStack

本章将讲述以下内容：

- ☐ 使用 cloud-init 运行安装后的命令
- ☐ 使用 cloud-config 运行安装后的配置
- ☐ 安装 OpenStack Telemetry
- ☐ 使用 OpenStack Telemetry 查看使用数据
- ☐ 安装 Neutron LBaaS
- ☐ 使用 Neutron LBaaS
- ☐ 配置 Neutron FWaaS
- ☐ 使用 Neutron FWaaS
- ☐ 安装 Heat OpenStack 编排服务
- ☐ 使用 Heat 启动实例

9.1 简介

到目前为止，本书探讨了如何构建并运营 OpenStack 云。由于本书需要，前面介绍了极为详细的环境搭建细节。在本章中，我们将学习一些 OpenStack 相关的项目、功能和特性，有助于提高有效涉及、实施、运营 OpenStack 云的能力。

9.2 使用 cloud-init 运行安装后的命令

cloud-init 原本由 Canonical 公司开发，是在云实例上运行安装后的命令与配置的标准工具。实例启动时，如果镜像中用到了 cloud-init，它会自动获取启动时传入的元数据，并

在安装完成后执行命令。如果使用了 shell 脚本(如本节中操作部分所示),就类似于在 Linux 机器的/etc/rc.local 工作目录下运行命令。cloud-init 需要 nova-metadata API 服务传输的数据才能运行。实例会查找与特点实例关联的数据,并根据情况执行命令。本节将介绍 cloud-init 的基础用法。

准备工作

确保登录到了可以访问 192.168.100.0/24 公网上 OpenStack 环境的 Ubuntu 主机。这台主机将用于运行 OpenStack 客户端工具。如果使用的是配套的 Vagrant 环境,可以使用 controller 节点,上面安装了提供 nova 命令行客户端的 python-novaclient 软件包。

如果使用 Vagrant 创建了该节点,可执行如下命令。

```
vagrant ssh controller
```

确保已经配置好了如下参数(如果用的不是 Vagrant 环境,请修改对应的证书路径和密钥文件路径)。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

本节将演示如何运行脚本唤起标准 Ubuntu 镜像的所有网络接口。如果不按本节的操作,以多个网络接口运行 Ubuntu 镜像时,只会唤起第一个接口 eth0。

本例中,我们将启动一个 Ubuntu 实例,创建两个 Neutron 网络,然后在命令行执行一个 shell 脚本将所有接口唤起。具体步骤如下。

1. 首先,创建一个 shell 脚本,演示向接口发送脚本的能力。在当前目录下创建一个名为 multi-nic.sh 的文件,写入以下内容。

```
#!/bin/bash
ifconfig -a | awk '/^eth/ {print $1}' | while read I
do
    dhclient $I
done
```

2. 然后将该文件作为参数,通过 -user-data 标志传递给 nova boot 命令。

```
nova boot
--flavor m1.tiny
--image trusty-image
--nic net-id=e8e4ed14-97a6-4715-a065-3ff0347f40dd
--nic net-id=8cb0f8f3-c529-45fe-ac5f-bd00c9814005
--user-data ./multi-nic.sh
myInstance
```

第二个 nic 和 eth1 现在会有一个关联的 IP。



net-id 的值可以通过运行 `neutron net-list` 命令查看，在输出在查找与想使用的网络相关的 ID 即可。可以指定任意数量的 `-nic` 标志（会受镜像能支持的最大数量限制）

工作原理

cloud-init 是一个非常强大的系统，是自动编排实例的基础。由于能够运行后安装脚本，cloud-init 为实现全栈系统自动化以及集成第三方配置管理工具奠定了基础。

为了能够执行类似 `rc.local` 的 cloud-init 脚本，文件的开头应为 `#!`，之后写入执行命令的解释器，如 `/bin/bash` 或 `/bin/python`。这会告诉 cloud-init 在初始服务都启动之后，再运行该脚本。

cloud-init 还支持其他许多特性，包括运行 `upstart` 作业（如果文件的开头为 `#upstart-job`，这会将文件放置到 `/etc/init` 目录下，与其他 `upstart` 作业一样执行），从而可以使用 `gzip` 文件。这些文件会被自动解压，向正常文件一样执行。由于输入大小被限制为 16,384 个字节，使用 `gzip` 文件是很常见的做法。

更多参考

更多有关 cloud-init 的信息，可查看 <http://cloud-init.readthedocs.org/>。

9.3 使用 cloud-init 运行安装后的配置

cloud-config 是 cloud-init 的一项特性，是通过 `apt` 安装软件包、配置实例的最简单的方式。通过 cloud-config，我们可以使用 YAML 文件描述如何配置实例，如果通过 shell 脚本进行就颇为麻烦。

准备工作

确保登录到了可以访问 192.168.100.0/24 公网上 OpenStack 环境的 Ubuntu 主机。这台主机将用于运行 OpenStack 客户端工具。如果使用的是配套的 Vagrant 环境，可以使用

controller 节点，上面安装了提供 nova 命令行客户端的 python-novaclient 软件包。

如果使用 Vagrant 创建了该节点，可执行如下命令。

```
vagrant ssh controller
```

确保已经配置好了如下参数（如果用的不是 Vagrant 环境，请修改对应的证书路径和密钥文件路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/akey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

本节将演示如何配置实例的主机名，安装 Apache Web 服务器，以及创建用户组和用户。具体步骤如下。

1. 首先，创建描述配置的 .yaml 文件。在当前目录下新建一个名为 webserver.yaml 的文件，写入如下内容。

```
#cloud-config
hostname: myWebserver
fqdn: mywebserver.cook.book
manage_etc_hosts: true
groups:
- developers
users:
- auser
  gecos: A User
  primary-group: users
  groups: users, developers
  passwd: $6r$j632wezy/grasdfds7/efew7fwq/fdfws.8ewfwefwe
packages
- apache2
```

2. 然后，将该文件作为参数，通过 -user-data 标志传递给 nova boot 命令。

```
nova boot
--flavor m1.tiny
--image trusty-image
--nic net-id=e8e4ed14-97a6-4715-a065-3ff0347f40dd
--user-data ./webserver.yaml
myWebserver
```



通过运行 neutron net-list 命令，可找到 net-id 的值，也能查找与想要使用的网络相关联的 id。

3. 输出可以在 nova console-log 中看到。clond-init 将输出如 apt-get update 和

apt-get install 之类的命令。在云配置运行的最后，你的 Web 服务器将是可访问的。



nova list 输出的内容显示了实例的状态，尽管 cloud-init 可能还没有完成，但它仍然会显示出 Active 的状态。如果依靠这个状态来检查运行在实例上的服务是否已准备好，请注意这一点。请记住，要为在实例上运行的服务打开相关的安全组端口。

工作原理

cloud-config 是 cloud-init 的一项特性，是在实例上执行安装后的配置的一种很简单的方法。可以配置不少选项。

在上面的例子中，我们设置了如下行。

```
#cloud-config
```

任何开头为该行内容的文件，都会被 cloud-init 视为 cloud-config 数据。

参数 hostname 设置主机的主机名。

参数 fqdn 设置主机的完全限定域名。

manage_etc_hosts: true 这一行允许用上面两个参数修改/etc/hosts 文件。

通过用户组和用户语句往实例中添加用户是很简单的。只需要列出要添加到系统的用户组 and 用户即可。每个用户的配置以下面该行开头。

```
- name: username
```

下一行是希望看到的系统信息。下一个用户的配置是这样开始的。

```
- name: anotheruser
```

Packages 设置部分负责要安装哪些软件包，我们只需列出希望安装的每个软件包即可，如下所示。如果软件包无法从本地网络获取，要确保实例可以连接到 Internet。

```
packages
- apache2
- openssl
```

如果无法访问外网，可以配置 apt 指向内部的软件仓库。

```
apt_mirror: http://internal-apt.cook.book/ubuntu/
apt_mirror_search_dns: false
```

该方法对于在无法访问 Internet 的实例上进行安装非常实用。

更多参考

Cloud-config 不止能够安装软件包，还可以用于安装、运行 Chef 和 Puppet 相关文件，支持 OpenStack 实例与第三方编排和配置管理系统进行集成。更多有关 cloud-config 的信息，

可查看 <http://cloud-init.readthedocs.org/>。

Cloud-init 不仅限于基于 Linux 的实例。Cloudbase 提供的 Cloudbase-init 为 Windows 实例提供了相同的能力。访问 <http://www.cloudbase.it/cloudinit-for-windows-instances/>，获取有关如何通过 Powershell 在 Windows 下设置 cloud-init 实例的更多信息。

9.4 安装 OpenStack Telemetry

OpenStack Telemetry 项目，即 Ceilometer，能够收集构成 OpenStack 组件的物理和虚拟资源的计量数据，并保存数据以备后续查找与分析。如果满足设置的条件，还可以自动触发特定的动作。

准备工作

确保已有运行 OpenStack 组件的合适的服务器。如果使用的是配套的 Vagrant 环境，可以使用同样的 controller 和 compute-01 节点来处理此类问题。

在 controller 和 compute 节点上安装 Ceilometer 程序包。确保可以在此环境中可登录到 controller 和 compute-01 节点。如果使用 Vagrant 创建了这些节点，可执行如下命令。

```
vagrant ssh controller
vagrant ssh compute-01
```

操作步骤

为了启用 Telemetry（即 ceilometer）服务，首先在控制节点上执行如下步骤。

1. Ceilometer 要求有一个专门的数据库存储它收集到的数据。将按照 MongoDB 和所需的依赖，用于 OpenStack Telemetry 服务。在控制节点上，执行如下命令。

```
sudo apt-get install mongodb python-pymongo python-bson
```

2. 安装好 MongoDB 之后，在控制节点上编辑 MongoDB 的配置文件 `/etc/mongodb.conf`，设置 `bind_ip` 参数。

```
bind_ip = 172.16.0.200
```

3. 重启 MongoDB。

```
sudo service mongodb restart
```

4. MongoDB 的命令使用 JavaScript 语法。为了配置 Ceilometer 使用 MongoDB，执行如下命令，添加 ceilometer 用户。


```
db.addUser( { user: "ceilometer",
               pwd: "openstack",
               roles: [ "readWrite", "dbAdmin" ]
             } );
```

5. Keystone 需要知道已经启用了 Ceilometer 服务, 所以执行如下命令, 确保 ceilometer 服务有 Keystone 权限。

```
keystone user-create --name=ceilometer --pass=ceilometer \
--email=ceilomoter@localhost
keystone user-role-add --user=ceilometer \
--tenant=service --role=admin
```

6. 然后, 执行以下命令, 在 Keystone 中为 Ceilometer 添加如下服务和端点。

```
keystone service-create --name=ceilometer \
--type=telemetry \
--description="Ceilometer Metering Service"
METERING_SERVICE_ID=$(keystone service-list \
| awk '/\ ceilometer/ {print $2}')
keystone endpoint-create \
--region RegionOne \
--service-id=${METERING_SERVICE_ID} \
--publicurl=http://192.168.100.200:8777 \
--internalurl=http://192.168.100.200:8777 \
--adminurl=http://192.168.100.200:8777
```

7. 现在可以使用 apt 安装所需的 ceilometer 软件包。

```
sudo apt-get update
sudo apt-install ceilometer-api \
ceilometer-collector \
ceilometer-agent-central \
python-ceilometerclient
```

8. 编辑/etc/ceilometer/ceilometer.conf 文件, 配置 ceilometer。文件中应包含如下配置, 才能正常使用。

```
[DEFAULT]
policy_file = /etc/ceilometer/policy.json
verbose = true
debug = true
insecure = true

##### AMQP #####
notification_topics = notifications, glance_notifications

rabbit_host=172.16.0.200
rabbit_port=5672
rabbit_userid=guest
rabbit_password=guest
rabbit_virtual_host=/
rabbit_ha_queues=false

[database]
```

```

connection=mongodb://ceilometer:openstack@172.16.0.200:27017/ceilometer

[api]
host = 172.16.0.200
port = 8777

[keystone_authtoken]
auth_uri = https://192.168.100.200:35357/v2.0/
identity_uri = https://192.168.100.200:5000
admin_tenant_name = service
admin_user = ceilometer
admin_password = ceilometer
revocation_cache_time = 10
insecure = True

[service_credentials]
os_auth_url = https://192.168.100.200:5000/v2.0
os_username = ceilometer
os_tenant_name = service
os_password = ceilometer
insecure = True

```

9. 重启所有 ceilometer 服务。

```

sudo service ceilometer-agent-central restart
sudo service ceilometer-agent-notification restart
sudo service ceilometer-alarm-evaluator restart
sudo service ceilometer-alarm-notifier restart
sudo service ceilometer-api restart
sudo service ceilometer-collector restart

```

配置好控制节点之后，接着在计算节点 compute-01 上安装 Ceilometer 代理。

10. 在计算节点 compute-01 上，安装 ceilometer 代理。

```
sudo apt-get install ceilometer-agent-compute
```

11. 编辑/etc/ceilometer/ceilometer.conf 文件，插入如下行。

```

[DEFAULT]
policy_file = /etc/ceilometer/policy.json
verbose = true
debug = true
insecure = true

##### AMQP #####
notification_topics = notifications,glance_notifications

rabbit_host=172.16.0.200
rabbit_port=5672
rabbit_userid=guest
rabbit_password=guest
rabbit_virtual_host=/
rabbit_ha_queues=false

```

```
[database]
connection=mongodb://ceilometer:openstack@172.16.0.200:27017/ceilometer

[api]
host = 172.16.0.200
port = 8777

[keystone_authtoken]
auth_uri = https://192.168.100.200:35357/v2.0/
identity_uri = https://192.168.100.200:5000
admin_tenant_name = service
admin_user = ceilometer
admin_password = ceilometer
revocation_cache_time = 10
insecure = True

[service_credentials]
os_auth_url = https://192.168.100.200:5000/v2.0
os_username = ceilometer
os_tenant_name = service
os_password = ceilometer
insecure = True
```

12. 将下面的 Ceilometer 部分配置，添加到/etc/nova/nova.conf 配置文件。

```
# Ceilometer
instance_usage_audit=True
instance_usage_audit_period=hour
notify_on_state_change=vm_and_task_state
notification_driver=nova.openstack.common.notifier.
rpc_notifier
```

13. 重启 nova 服务。

```
sudo service nova-compute restart
```

14. 重启 Ceilometer 代理。

```
sudo service ceilometer-agent-compute restart
```

compute 节点现在会将使用数据，报告给 controller 节点。

工作原理

我们在 controller 节点上安装并配置了 MongoDB，设置 MongoDB 监听内部控制节点的 IP，类似一个 RabbitMQ 服务。Ceilometer 通过 keystone 进行通信，所以使用 keystone 命令创建了 ceilometer 用户和服务。之后，我们安装并配置了 ceilometer 软件包。Ceilometer 的配置文件是/etc/ceilometer/ceilometer.conf。除了[api]、[keystone_token]和[service_credentials]部分外，还设置了数据库连接参数。

```
[database]
```

```
connection=mongodb://ceilometer:openstack@172.16.0.200:27017/ceilometer
```




有关如何正确在生产环境配置 MongoDB，请参考 MongoDB 官方文档 <http://docs.mongodb.org/manual/>。

我们可以使用其他任何 Ceilometer 支持的数据库，如 MongoDB、MySQL、PostgreSQL、HBase 和 DB2。

配置好控制节点后，我们在计算节点 compute-01 上安装了 Ceilometer 代理。然后更新了 `/etc/ceilometer.conf` 和 `/etc/nova/nova.conf` 文件。至此，Ceilometer 服务配置完毕。

9.5 使用 OpenStack Telemetry 查看使用数据

安装并配置好 Telemetry 模块之后，现在可以使用 Ceilometer 命令行查看资源使用数据。通过获取环境中配置的指标，列出数据，即可查看。

准备工作

确保登录到了可以访问 192.168.100.0/24 公网上 OpenStack 环境的 Ubuntu 主机。这台主机将用于运行 OpenStack 客户端工具。如果使用的是配套的 Vagrant 环境，可以使用 controller 节点，上面安装了提供 ceilometer 命令行客户端的 `python-novaclient` 软件包。

如果使用 Vagrant 创建了该节点，可执行如下命令。

```
vagrant ssh controller
```

确保已经配置好了如下参数（如果用的不是 Vagrant 环境，请修改对应的证书路径和密钥文件路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/akey.pem
export OS_CACERT=/vagrant/ca.pem
vagrant ssh controller
```

操作步骤

首先列出环境中可用的指标。

1. 先列出环境中可用的指标。

```
ceilometer meter-list
```

输出如图 9-1 所示。

Name	Type	Unit	Resource ID	User ID	Project ID
cpu	cumulative	ns	4be430e2-9b12	37e4ba2f	d27f89204b8
cpu_util	gauge	%	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.device.read.bytes	cumulative	B	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.device.read.requests	cumulative	request	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.device.write.bytes	cumulative	B	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.device.write.requests	cumulative	request	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.read.bytes	cumulative	B	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.read.bytes.rate	gauge	B/s	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.read.requests	cumulative	request	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.read.requests.rate	gauge	request/s	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.write.bytes	cumulative	B	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.write.bytes.rate	gauge	B/s	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.write.requests	cumulative	request	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.write.requests.rate	gauge	request/s	4be430e2-9b12	37e4ba2f	d27f89204b8
image	gauge	image	b8ab5287-081a	None	d27f89204b8
image	gauge	image	e5f33281-c497	None	d27f89204b8
image.size	gauge	B	b8ab5287-081a	None	d27f89204b8
image.size	gauge	B	e5f33281-c497	None	d27f89204b8
instance	gauge	instance	4be430e2-9b12	37e4ba2f	d27f89204b8
instance:m1.tiny	gauge	instance	4be430e2-9b12	37e4ba2f	d27f89204b8
network.incoming.bytes	cumulative	B	instance-0000	37e4ba2f	d27f89204b8
network.incoming.bytes.rate	gauge	B/s	instance-0000	37e4ba2f	d27f89204b8
network.incoming.packets	cumulative	packet	instance-0000	37e4ba2f	d27f89204b8
network.incoming.packets.rate	gauge	packet/s	instance-0000	37e4ba2f	d27f89204b8
network.outgoing.bytes	cumulative	B	instance-0000	37e4ba2f	d27f89204b8
network.outgoing.bytes.rate	gauge	B/s	instance-0000	37e4ba2f	d27f89204b8
network.outgoing.packets	cumulative	packet	instance-0000	37e4ba2f	d27f89204b8
network.outgoing.packets.rate	gauge	packet/s	instance-0000	37e4ba2f	d27f89204b8

图 9-1

2. 指定虚拟机来降低指标的数量：

ceilometer meter-list --query resource=4be430e2-9b12

输出如图 9-2 所示。

Name	Type	Unit	Resource ID	User ID	Project ID
cpu	cumulative	ns	4be430e2-9b12	37e4ba2f	d27f89204b8
cpu_util	gauge	%	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.read.bytes	cumulative	B	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.read.bytes.rate	gauge	B/s	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.read.requests	cumulative	request	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.read.requests.rate	gauge	request/s	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.write.bytes	cumulative	B	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.write.bytes.rate	gauge	B/s	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.write.requests	cumulative	request	4be430e2-9b12	37e4ba2f	d27f89204b8
disk.write.requests.rate	gauge	request/s	4be430e2-9b12	37e4ba2f	d27f89204b8
instance	gauge	instance	4be430e2-9b12	37e4ba2f	d27f89204b8
instance:m1.tiny	gauge	instance	4be430e2-9b12	37e4ba2f	d27f89204b8

图 9-2

3. 我们可以查看虚拟机相关指标的示例数据。使用如下命令即可：

ceilometer sample-list -m disk.write.bytes

输出如图 9-3 所示，包含许多原始数据。

Resource ID	Name	Type	Volume	Unit	Timestamp
4be430e2-9b12-4e1b-941c-58b5d0f0918b	disk.write.bytes	cumulative	707003392.0	B	2015-04-20T08:48:16
4be430e2-9b12-4e1b-941c-58b5d0f0918b	disk.write.bytes	cumulative	707003392.0	B	2015-04-20T08:38:16
4be430e2-9b12-4e1b-941c-58b5d0f0918b	disk.write.bytes	cumulative	707003392.0	B	2015-04-20T08:28:16
4be430e2-9b12-4e1b-941c-58b5d0f0918b	disk.write.bytes	cumulative	707003392.0	B	2015-04-20T08:18:16
4be430e2-9b12-4e1b-941c-58b5d0f0918b	disk.write.bytes	cumulative	706917376.0	B	2015-04-20T08:08:16
4be430e2-9b12-4e1b-941c-58b5d0f0918b	disk.write.bytes	cumulative	706917376.0	B	2015-04-20T07:58:16
4be430e2-9b12-4e1b-941c-58b5d0f0918b	disk.write.bytes	cumulative	706917376.0	B	2015-04-20T07:48:16
4be430e2-9b12-4e1b-941c-58b5d0f0918b	disk.write.bytes	cumulative	706917376.0	B	2015-04-20T07:38:16
4be430e2-9b12-4e1b-941c-58b5d0f0918b	disk.write.bytes	cumulative	706917376.0	B	2015-04-20T07:28:16
4be430e2-9b12-4e1b-941c-58b5d0f0918b	disk.write.bytes	cumulative	706917376.0	B	2015-04-20T07:18:16
4be430e2-9b12-4e1b-941c-58b5d0f0918b	disk.write.bytes	cumulative	706810880.0	B	2015-04-20T07:08:16

图 9-3

为了方便阅读，这里裁剪了上述输出；一般来说输出中的数据量会更多。

4. Ceilometer 提供了一些实用工具，用于执行计算，提供数据。可以通过如下命令获取单个指标的数据，如查看 disk write 的速率：

```
ceilometer statistics --meter disk.write.bytes.rate
```

输出如图 9-4 所示。

Period	Period Start	Period End	Max	Min	Avg	Sum	Count	Duration	Duration Start	Duration End
0	2015-04-14T03:58:15	2015-04-14T03:58:15	134778.88	0.0	1329.6	1163356.4	875	529201.0	2015-04-14T03:58:15	2015-04-20T06:58:16

图 9-4

5. Ceilometer 支持自定义查询。输入起始时间，即可查看某个时间段的数据：

```
ceilometer statistics -m disk.write.bytes.rate \
-q 'timestamp>2015-04-14T22:38:15; \
timestamp<=2015-04-19T04:28:15' --period 60
```

上述命令会输出这段时间内的所有数据。下面是裁剪后的输出，如图 9-5 所示。

Period	Period Start	Period End	Max	Min	Avg	Sum	Count	Duration	Duration Start	Duration End
60	2015-04-14T22:48:15	2015-04-14T22:49:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-14T22:48:15	2015-04-14T22:48:15
60	2015-04-14T22:58:15	2015-04-14T22:59:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-14T22:58:15	2015-04-14T22:58:15
60	2015-04-14T23:08:15	2015-04-14T23:09:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-14T23:08:15	2015-04-14T23:08:15
60	2015-04-14T23:18:15	2015-04-14T23:19:15	177.4	177.4	177.4	177.4	1	0.0	2015-04-14T23:18:15	2015-04-14T23:18:15
60	2015-04-14T23:28:15	2015-04-14T23:29:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-14T23:28:15	2015-04-14T23:28:15
60	2015-04-14T23:38:15	2015-04-14T23:39:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-14T23:38:15	2015-04-14T23:38:15
60	2015-04-14T23:48:15	2015-04-14T23:49:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-14T23:48:15	2015-04-14T23:48:15
60	2015-04-14T23:58:15	2015-04-14T23:59:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-14T23:58:15	2015-04-14T23:58:15
60	2015-04-15T00:08:15	2015-04-15T00:09:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T00:08:15	2015-04-15T00:08:15
60	2015-04-15T00:18:15	2015-04-15T00:19:15	129.7	129.7	129.7	129.7	1	0.0	2015-04-15T00:18:15	2015-04-15T00:18:15
60	2015-04-15T00:28:15	2015-04-15T00:29:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T00:28:15	2015-04-15T00:28:15
60	2015-04-15T00:38:15	2015-04-15T00:39:15	61.44	61.44	61.44	61.44	1	0.0	2015-04-15T00:38:15	2015-04-15T00:38:15
60	2015-04-15T00:48:15	2015-04-15T00:49:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T00:48:15	2015-04-15T00:48:15
60	2015-04-15T00:58:15	2015-04-15T00:59:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T00:58:15	2015-04-15T00:58:15
60	2015-04-15T01:08:15	2015-04-15T01:09:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T01:08:15	2015-04-15T01:08:15
60	2015-04-15T01:18:15	2015-04-15T01:19:15	129.7	129.7	129.7	129.7	1	0.0	2015-04-15T01:18:15	2015-04-15T01:18:15
60	2015-04-15T01:28:15	2015-04-15T01:29:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T01:28:15	2015-04-15T01:28:15
60	2015-04-15T01:38:15	2015-04-15T01:39:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T01:38:15	2015-04-15T01:38:15
60	2015-04-15T01:48:15	2015-04-15T01:49:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T01:48:15	2015-04-15T01:48:15
60	2015-04-15T01:58:15	2015-04-15T01:59:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T01:58:15	2015-04-15T01:58:15
60	2015-04-15T02:08:15	2015-04-15T02:09:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T02:08:15	2015-04-15T02:08:15
60	2015-04-15T02:18:15	2015-04-15T02:19:15	129.7	129.7	129.7	129.7	1	0.0	2015-04-15T02:18:15	2015-04-15T02:18:15
60	2015-04-15T02:28:15	2015-04-15T02:29:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T02:28:15	2015-04-15T02:28:15
60	2015-04-15T02:38:15	2015-04-15T02:39:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T02:38:15	2015-04-15T02:38:15
60	2015-04-15T02:48:15	2015-04-15T02:49:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T02:48:15	2015-04-15T02:48:15
60	2015-04-15T02:58:15	2015-04-15T02:59:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T02:58:15	2015-04-15T02:58:15
60	2015-04-15T03:08:15	2015-04-15T03:09:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T03:08:15	2015-04-15T03:08:15
60	2015-04-15T03:18:15	2015-04-15T03:19:15	129.7	129.7	129.7	129.7	1	0.0	2015-04-15T03:18:15	2015-04-15T03:18:15
60	2015-04-15T03:28:15	2015-04-15T03:29:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T03:28:15	2015-04-15T03:28:15
60	2015-04-15T03:38:15	2015-04-15T03:39:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T03:38:15	2015-04-15T03:38:15
60	2015-04-15T03:48:15	2015-04-15T03:49:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T03:48:15	2015-04-15T03:48:15
60	2015-04-15T03:58:15	2015-04-15T03:59:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T03:58:15	2015-04-15T03:58:15
60	2015-04-15T04:08:15	2015-04-15T04:09:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T04:08:15	2015-04-15T04:08:15
60	2015-04-15T04:18:15	2015-04-15T04:19:15	402.7	402.7	402.7	402.7	1	0.0	2015-04-15T04:18:15	2015-04-15T04:18:15
60	2015-04-15T04:28:15	2015-04-15T04:29:15	0.0	0.0	0.0	0.0	1	0.0	2015-04-15T04:28:15	2015-04-15T04:28:15

图 9-5

6. 如需调查累积数据的统计信息，可通过如下命令检查网络对外流量（字节数）：

```
ceilometer statistics -m network.outgoing.bytes
```

输出如图 9-6 所示。

Period	Period Start	Period End	Max	Min	Avg
0	2015-04-14T03:48:15	2015-04-14T03:48:15	56915.0	25157.0	41814.138009

Sum	Count	Duration	Duration Start	Duration End
36963698.0	884	534601.0	2015-04-14T03:48:15	2015-04-20T08:18:16

图 9-6



这里，我们处理了下表格输出以方便阅读。正常来说，所有的列是一起呈现的。

主要的区别在于，除了之前的所有其列之外，现在还有一个具有所有传出字节总和的 Sum 列。

7. 使用对特定时间段的查询，列出网络对外流量的总量：

```
ceilometer statistics -m network.outgoing.bytes \
-q 'timestamp>2015-04-14T22:38:15;\
timestamp<=2015-04-19T04:28:15' --period 60
```

输出如图 9-7 所示。

Period Start	Period End	Max	Min	Avg	Sum	Duration Start	Duration End
2015-04-14T22:48:15	2015-04-14T22:49:15	31323.0	31323.0	31323.0	31323.0	2015-04-14T22:48:15	2015-04-14T22:48:15
2015-04-14T22:58:15	2015-04-14T22:59:15	31323.0	31323.0	31323.0	31323.0	2015-04-14T22:58:15	2015-04-14T22:58:15
2015-04-14T23:08:15	2015-04-14T23:09:15	31323.0	31323.0	31323.0	31323.0	2015-04-14T23:08:15	2015-04-14T23:08:15
2015-04-14T23:18:15	2015-04-14T23:19:15	31323.0	31323.0	31323.0	31323.0	2015-04-14T23:18:15	2015-04-14T23:18:15
2015-04-14T23:28:15	2015-04-14T23:29:15	31323.0	31323.0	31323.0	31323.0	2015-04-14T23:28:15	2015-04-14T23:28:15
2015-04-14T23:38:15	2015-04-14T23:39:15	31323.0	31323.0	31323.0	31323.0	2015-04-14T23:38:15	2015-04-14T23:38:15
2015-04-14T23:48:15	2015-04-14T23:49:15	31323.0	31323.0	31323.0	31323.0	2015-04-14T23:48:15	2015-04-14T23:48:15
2015-04-14T23:58:15	2015-04-14T23:59:15	31323.0	31323.0	31323.0	31323.0	2015-04-14T23:58:15	2015-04-14T23:58:15
2015-04-15T00:08:15	2015-04-15T00:09:15	31323.0	31323.0	31323.0	31323.0	2015-04-15T00:08:15	2015-04-15T00:08:15
2015-04-15T00:18:15	2015-04-15T00:19:15	31323.0	31323.0	31323.0	31323.0	2015-04-15T00:18:15	2015-04-15T00:18:15
2015-04-15T00:28:15	2015-04-15T00:29:15	31323.0	31323.0	31323.0	31323.0	2015-04-15T00:28:15	2015-04-15T00:28:15
2015-04-15T00:38:15	2015-04-15T00:39:15	31707.0	31707.0	31707.0	31707.0	2015-04-15T00:38:15	2015-04-15T00:38:15
...							
2015-04-19T03:58:15	2015-04-19T03:59:15	48345.0	48345.0	48345.0	48345.0	2015-04-19T03:58:16	2015-04-19T03:58:16
2015-04-19T04:08:15	2015-04-19T04:09:15	48345.0	48345.0	48345.0	48345.0	2015-04-19T04:08:16	2015-04-19T04:08:16
2015-04-19T04:18:15	2015-04-19T04:19:15	48345.0	48345.0	48345.0	48345.0	2015-04-19T04:18:16	2015-04-19T04:18:16

图 9-7

这里可以看到指定时间段的累积流量之和。为了方便阅读，我们移除了 Period、Count、Duration 列，以及中间的一些数据。

工作原理

Ceilometer 从多个 OpenStack 组件收集数据。我们查看了环境中配置的统计指标，查询了具体时间段内的数据。Ceilometer 提供三类数据：累计值（cumulative）、离散值（gauge）

和变化值 (delta)。我们查看了离散数据, 包含了磁盘写速率的信息。还查看了某个时间段的累计数据。可使用如下命令, 列出所有的 Ceilometer 指标。

```
ceilometer meter-list
```

可使用如下命令, 提供 -m 或 -meter 标志, 即可查看单个指标的示例数据。

```
ceilometer sample-list -m disk.write.bytes
```

Ceilometer 还提供了各种数据的统计信息。可使用如下命令, 查看统计数据。

```
ceilometer statistics --meter disk.write.bytes.rate
```

上述命令还接受以下格式的查询。

```
ceilometer statistics -m disk.write.bytes.rate -q \
'timestamp>2015-04-14T22:38:15;\
timestamp<=2015-04-19T04:28:15' --period 60
```



根据主机和指标数量的不同, Ceilometer 收集、存储和查询的数据, 可以是中等或大规模的数据, 或大数据。虽然介绍如何处理这类数据问题超出了本书的范围, 但是建议如果将 Ceilometer 部署到生产环境, 应该将其部署在不会影响到工作负载的地方。

9.6 安装 Neutron LBaaS

OpenStack Neutron 拥有可扩展的插件机制, 可通过 Neutron API 实现更多的网络特性。通过在网络节点上启用负载均衡即服务 (Load-Balancer-as-a-Service, 即 LBaaS), 我们可以通过调用 Neutron API 创建和管理负载均衡器。该机制支持多个硬件厂商; 以下示例使用的是 Open vSwitch 的 HA Proxy 参考驱动。

我们在网络节点上安装 LBaaS 代理, 并在网络节点和控制节点上配置 Neutron 使得服务设置生效, 即可使用 Neutron LBaaS。

准备工作

确保有一台运行 OpenStack 网络组件的服务器。如果你使用的是配套 Vagrant 环境, 可使用前言中提到的网络节点。

确保登录到环境中的网络节点和控制节点。如果通过 Vagrant 创建了这两个节点, 可执行如下命令。

```
vagrant ssh network
vagrant ssh controller
```

操作步骤


在网络节点上执行如下步骤，启用 Neutron 的 LBaaS 特性。

1. 使用 apt 安装 LBaaS 代理。

```
sudo apt-get update
sudo add-apt-repository ppa:openstack-ubuntu-testing/kilo
sudo apt-get install neutron-lbaas-agent haproxy
```

2. 在 `/etc/neutron/neutron.conf` 文件的 `[DEFAULT]` 部分，添加 `lbaas` 到 `service_plugins` 一行，启用负载均衡服务。

```
service_plugins = lbaas
```

 `service_plugin` 这行是一个逗号分隔的列表，如：`service_plugins = lbaas, router.`

3. 在 `/etc/neutron/neutron.conf` 文件中添加如下行设置，启用 HA Proxy 负载均衡服务。

```
[service_providers]
service_provider =
LOADBALANCER:Haproxy:neutron.services.loadbalancer.drivers.
haproxy.plugin_driver.HaproxyOnHostPluginDriver:default
```

4. 然后，编辑 `/etc/neutron/lbaas_agent.ini` 文件，加入如下行。

```
[DEFAULT]
debug = False
interface_driver =
neutron.agent.linux.interface.OVSInterfaceDriver
device_driver =
neutron.services.loadbalancer.drivers.haproxy.
namespace_driver.HaproxyNSDriver
```


```
[haproxy]
loadbalancer_state_path = $state_path/lbaas
user_group = nogroup
```

5. 在网络节点上启动 Neutron LBaaS 代理。

```
sudo start neutron-lbaas-agent
```

6. 在运行 Neutron API 的控制节点上，编辑 `/etc/neutron/neutron.conf` 文件，具体内容与上面提到的配置相同。在 `[DEFAULT]` 部分下，往 `service_plugins` 这行添加 `lbaas`。

```
service_plugins = lbaas
```

 `service_plugin` 这行是一个逗号分隔的列表，如：`service_plugins = lbaas, router.`

7. 然后, 向/etc/neutron/neutron.conf 配置文件中添加如下行, 启动 HA Proxy 负载均衡服务。

```
[service_providers]
service_provider =
LOADBALANCER:Haproxy:neutron.services.loadbalancer.drivers.
haproxy.plugin_driver.HaproxyOnHostPluginDriver:default
```

8. 在控制节点上, 重启 Neutron API 服务。

```
sudo restart neutron-server
```

9. 可以配置 OpenStack Dashboard, 即 Horizon, 来试用 LBaaS。如需启用该功能, 编辑/etc/openstack-dashboard/local_settings.py 文件, 加入如下行。

```
OPENSTACK_NEUTRON_NETWORK = {
    'enable_lb': True,
    ...
}
```

10. 重启 Apache 服务器, 使得变动生效。

```
service apache2 restart
```

工作原理

在环境中启用 Neutron LBaaS 插件的步骤大致包括: 首先在网络节点上安装 LBaaS 代理软件包, 然后配置代理试用 HA Proxy。/etc/neutron/neutron.conf 文件中加入如下行设置之后, 会告知 neutron 服务已启用该服务。

```
[Default]
service_plugins = lbaas
[Service_Providers]
service_provider =
LOADBALANCER:Haproxy:neutron.services.loadbalancer.drivers.haproxy
.plugin_driver.HaproxyOnHostPluginDriver:default
```

LBaaS 代理的具体配置, 通过网络节点上的/etc/neutron/lbaas-agent.ini 文件来实现。

然后通知控制节点上运行的 Neutron API 服务, 告知已成功安装 Neutron LBaaS 插件。接着, 将之前创建的 neutron.conf 文件复制到控制节点, 并重启 Neutron API 服务。

9.7 使用 Neutron LBaaS

安装好 Neutron LBaaS 之后, 可以通过 Neutron API 和命令行使用。这样支持为实例创建简单的 HA Proxy 负载均衡服务, 通过创建负载均衡池并将运行的实例添加到这些池即可

实现。另外，还可以加入监控功能，帮助负载均衡器判断是否发送流量到某个实例。

本节将配置一个基本的 HTTP 负载均衡池，加入两个运行 Apache 的服务器。我们可以使用负载均衡池，向这两个实例发送流量。

准备工作

确保登录到了可以访问 192.168.100.0/24 公网上 OpenStack 环境的 Ubuntu 主机。这台主机将用于运行 OpenStack 客户端工具。如果使用的是配套的 Vagrant 环境，可以使用 controller 节点，上面安装了提供 neutron 命令行客户端的 python-novaclient 软件包。

如果使用 Vagrant 创建了该节点，可执行如下命令。

```
vagrant ssh controller
```

确保已经配置好了如下参数（如果用的不是 Vagrant 环境，请修改对应的证书路径和密钥文件路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

首先创建负载均衡池，然后加入运行 Apache 的成员实例，具体步骤如下。

1. 首先，列出环境中可用的子网，选择在哪个子网上创建负载均衡器，如图 9-8 所示。

```
neutron subnet-list
```

id	name	cidr	allocation_pools
11c11dca-479b-435d-889d-fc29479b0a24	cookbook_float_subnet_1	192.168.100.0/24	{"start": "192.168.100.10", "end": "192.168.100.20"}
4cf2c09c-b3d5-40ed-9127-ec40e5e38343	cookbook_subnet_1	10.200.0.0/24	{"start": "10.200.0.2", "end": "10.200.0.254"}

图 9-8

2. 可通过希望使用的子网的 subnet-id 值，创建负载均衡池。本示例使用其中的一个私有子网 cookbook_subnet_1:

```
neutron lb-pool-create \
  --description "Web Load Balancer" \
  --lb-method ROUND_ROBIN \
  --name Web-Load-Balancer \
  --protocol HTTP \
  --subnet-id 11c11dca-479b-435d-889d-fc29479b0a24
```

上述命令的输出如图 9-9 所示。

Created a new pool:

Field	Value
admin_state_up	True
description	Web Load Balancer
health_monitors	
health_monitors_status	
id	546e1f04-b059-453e-a619-c0e4efff8a52
lb_method	ROUND_ROBIN
members	
name	Web-Load-Balancer
protocol	HTTP
provider	haproxy
status	PENDING_CREATE
status_description	
subnet_id	624ff352-2bca-40a2-bb45-efe0b824a6a1
tenant_id	61fbd985a834dce8f68f8dfabd73bd0
vip_id	

图 9-9

3. 现在将实例添加到该负载均衡池。首先确保这两个实例有运行 Apache，可参考 9.2 节中的步骤，在相同子网上创建符合要求的服务器。

```
nova boot \
--flavor m1.tiny \
--image trusty-image \
--nic net-id=25153759-994f-4835-9b13-bf0ec77fb336 \
--user-data ./webserver.yaml \
--max-count 2 \
webServer
```

4. 这两台 Web 服务器运行之后，获得赋予它们的 IP 地址，并制定给之前创建的负载均衡池 Web-Load-Balancer:

```
nova list
```

上述命令的输出如图 9-10 所示。

ID	Name	Status	Task State	Power State	Networks
70672037-cc50-45b1-090e-5cedb606552c	test1	ACTIVE	-	Running	cookbook_network_1=11.200.0.2, 192.168.100.11
8ab5d1ed-7c17-45ca-9d41-daf48e27af20	webserver-8ab5d1ed-7c17-45ca-9d41-daf48e27af20	ACTIVE	-	Running	cookbook_network_1=11.200.0.5
3d1eba44-25a1-4aa9-8ce9-005531dbd5e1	webserver-3d1eba44-25a1-4aa9-8ce9-005531dbd5e1	ACTIVE	-	Running	cookbook_network_1=11.200.0.6

图 9-10

5. 首先将第一台服务器的 IP 10.200.0.4 添加到负载均衡池:

```
neutron lb-member-create \
--address 10.200.0.4 \
--protocol-port 80 \
Web-Load-Balancer
```

上述命令的输出如图 9-11 所示。

6. 对第二台服务器的 IP 执行相同操作。

```
neutron lb-member-create \
--address 10.200.0.5 \
--protocol-port 80 \
Web-Load-Balancer
```


Created a new member:

Field	Value
address	10.200.0.4
admin_state_up	True
id	5cade31f-7208-4623-88d2-f0d60818cce5
pool_id	7d1aef98-90ee-4b6f-82a2-1b4b39c3c444
protocol_port	80
status	PENDING_CREATE
status_description	
tenant_id	001504d6f08d4698824636fa00f4a0f2
weight	1

图 9-11

7. 然后, 通过如下命令查看负载均衡池成员的状态:

```
neutron lb-member-list
```

上述命令的输出如图 9-12 所示。

id	address	protocol_port	weight	admin_state_up	status
af96b7f1-e2b0-4f5c-ae70-7e99a75079bf	11.200.0.5	80	1	True	ACTIVE
eelaa038-53d5-44df-ae29-89af7548d9fc	11.200.0.6	80	1	True	ACTIVE

图 9-12

8. 现在需要创建虚拟 IP (VIP), 用于访问负载均衡池及其中的实例。这个 VIP 要创建在公共网络上:

```
neutron subnet-list
```

上述命令的输出如图 9-13 所示。

id	name	cidr	allocation_pools
11c11dca-479b-435d-889d-fc29479b0a24	cookbook_float_subnet_1	192.168.100.0/24	{"start": "192.168.100.10", "end": "192.168.100.20"}
4cf2c09c-b3d5-40ed-9127-ec40e5e38343	cookbook_subnet_1	10.200.0.0/24	{"start": "10.200.0.2", "end": "10.200.0.254"}

图 9-13

9. 接着使用浮动/外部子网 ID, 创建可公开访问的 VIP:

```
neutron lb-vip-create \
  --name WebserverVIF \
  --protocol-port 80 \
  --protocol HTTP \
  --subnet-id 11c11dca-479b-435d-889d-fc29479b0a24 \
  Web-Load-Balancer
```

上述命令的输出如图 9-14 所示。

新创建的 IP 地址为 192.168.100.12, 可使用网络浏览器打开该地址; 负载均衡会均衡两台 Web 服务器之间的流量。

Created a new vip:

Field	Value
address	192.168.100.12
admin_state_up	True
connection_limit	-1
description	
id	c66c59a8-b34a-47e2-8239-63aa685c96c5
name	WebserverVIP
pool_id	7d1aef98-90ee-4b6f-82a2-1b4b39c3c444
port_id	1b1f227c-9439-47fc-b6f9-e0c24e03b5b9
protocol	HTTP
protocol_port	80
session_persistence	
status	PENDING_CREATE
status_description	
subnet_id	11c11dca-479b-435d-889d-fc29479b0a24
tenant_id	001504d6f08d4698824636fa00f4a0f2

图 9-14

工作原理

我们创建一个带有两台实例的负载均衡池，然后为其指定了一个 VIP，用于访问实例。具体需要执行如下步骤。

1. 通过 `neutron lb-pool-create` 命令，创建负载均衡池。
2. 通过 `neutron lb-member create` 命令，加入所使用的子网 IP，从而向负载均衡池添加成员实例。
3. 通过 `neutron lb-vip create` 命令，在外部浮动 IP 段上创建一个 VIP，使得公网可以访问负载均衡。
4. 创建负载均衡池时，使用如下语法。

```
neutron lb-pool-create \
  --description $DESCRIPTION \
  --lb-method $LB_METHOD \
  --name $LB_NAME \
  --protocol $PROTOCOL \
  --subnet-id $SUBNET_ID
```

9.8 配置 Neutron FWaaS

配置好 OpenStack Neutron LBaaS 插件之后，我们再来了解另一个有用的插件 FWaaS (FireWall as a Service, 防火墙即服务)。在网络节点启用 FWaaS 代理之后，将能够通过 Neutron API 创建和管理防火墙。不同的硬件要求的驱动也不同；本节的示例使用 IPTables

来提供防火墙服务。

我们在运行 Neutron L3 代理的节点（如果没有使用分布式虚拟路由 DVR 的话，这个节点将是在网络节点）或计算节点（如果使用了 DVR）上配置 Neutron FWaaS，并在控制节点上配置 Neutron Server API，使得之前的配置生效。还可以在 Horizon 界面中启用 FWaaS 功能。

准备工作

确保有一台运行 OpenStack 网络组件的服务器。如果你使用的是配套 Vagrant 环境，可使用前言中提到的网络节点。

确保登录到环境中的网络节点和控制节点。如果通过 Vagrant 创建了这两个节点，可执行如下命令。


```
vagrant ssh network
vagrant ssh controller
```

操作步骤

为了启用 Neutron FWaaS 功能，首先在运行 L3 代理的节点上执行如下步骤。在正常情况下，这个节点是网络节点。如果运行有 DVR，这个节点就是计算节点，具体步骤如下。

1. 启用防火墙服务。在 `/etc/neutron/neutron.conf` 文件的 `[DEFAULT]` 部分，向 `service_plugins` 这行加入 `firewall`。

```
service_plugins = firewall
```

 `service_plugin` 这行是一个逗号分隔的列表，如：`service_plugins = router, firewall.`

2. 在同一个文件中，向 `[SERVICE_PROVIDERS]` 部分加入如下行。

```
[SERVICE_PROVIDERS]
service_provider =
FIREWALL:Iptables:neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver:default
```

3. 然后编辑 `/etc/neutron/fwaas_driver.ini`，加入如下设置。

```
[fwaas]
driver = neutron.services.firewall.drivers.linux.iptables_fwaas.
IptablesFwaasDriver
enabled = True
```

4. 重启 `neutron-l3-agent` 服务，使得上述改动生效。

```
sudo restart neutron-l3-agent
```

5. 在运行 Neutron Server API 和 Horizon 的控制节点上，对 `/etc/neutron/neturon.`

conf 文件做相同的修改。

```
[DEFAULT]
service_plugins = firewall

[SERVICE_PROVIDERS]
service_provider =
FIREWALL:Iptables:neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver:default
```



如果 service_plugin 这行已经有其他内容，就将 firewall 添加在后面，如：service_plugins = router, firewall。

6. 重启 Neutron Server 服务，使得改动生效。

```
sudo restart neutron-server
```

7. 编辑/etc/openstack-dashboard/local_settings.py 文件，在 Horizon 中启用 FWaaS 功能。

```
OPENSTACK_NEUTRON_NETWORK = {
    'enable_firewall': True,
    ...
}
```

8. 重启 Apache，使得改动生效。

```
sudo service apache2 restart
```

工作原理

通过在运行 L3 代理的节点上（非 DVR 模式为网络节点，DVR 模式则为计算节点）配置相关的 Neutron 配置文件，我们成功在环境中启用了 Neutron FWaaS 插件。

在/etc/neutron/neutron.conf 文件中加入了如下设置，告知 Neutron 服务已启用该功能。

```
[DEFAULT]
service_plugins = firewall

[SERVICE_PROVIDERS]
service_provider =
FIREWALL:Iptables:neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver:default
```

FWaaS 代理的具体配置，通过在运行 L3 代理的节点上编辑/etc/neutron/fwaas-driver.ini 文件实现。

然后告知控制节点上运行的 Neutron API 服务已经启用新功能。复制此前的 neutron.conf 设置到控制节点上，然后重启 Neutron Server API 服务。

最后, 将 `enable_firewall` 的值设置为 `True`, 并重启 `Apache` 使得改动生效, 这样就在 `Horizon` 中启用了该功能。

9.9 使用 Neutron FWaaS

安装好 `Neutron FWaaS` 之后, 可以通过 `Neutron API` 和命令行使用该服务, 这支持在不同的 `Neutron` 网络之间建立防火墙规则。

在 `L3` 路由器上建立 `Neutron` 防火墙后, 任何途径该路由器的流量在继续之前, 都会受到审查。这样, 我们可以在应用的不同层级之间建立起防火墙。例如, 在一个标准的多层 `Web` 应用中, `Web` 服务器需与数据库服务器通信。有了防火墙, 就可以做到只允许数据库和 `Web` 服务器之间发生数据库流量。路由器级别的规则可以看作传统的边缘防火墙规则, 而安全组则类似基于主机的安全规则。策略驱动的安全, 也适用于向 `OpenStack` 环境转变的传统网络安全团队, 可以实现网络层级而不是计算层级的标准管控。

图 9-15 清楚地解释了 `Neutron FWaaS` 带来的服务分隔情况。在本例中, `Web` 应用层与数据库层处于不同的子网, `Neutron L3` 路由器和 `FWaaS` 负责转发二者之间的流量。这里适用的规则允许数据库和 `Web` 服务器之间通信——如只允许 `Web` 应用层子网的 3306 `TCP` 端口连接到数据库层的子网。

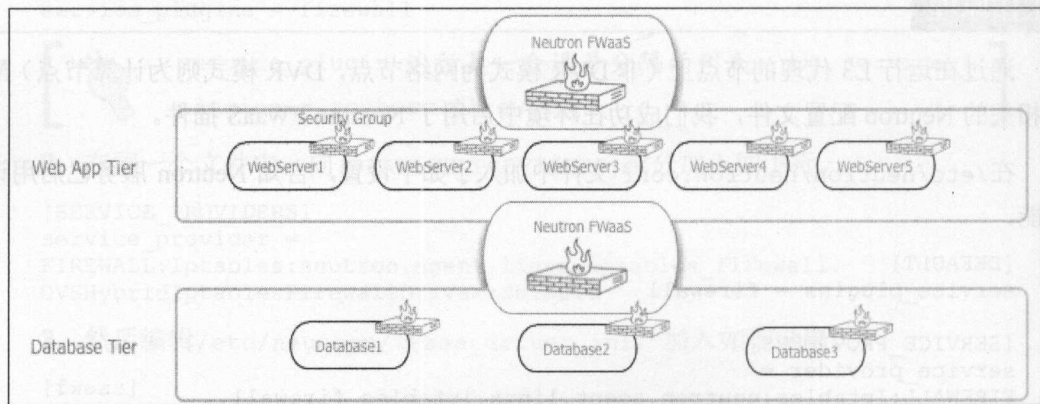


图 9-15

准备工作

确保登录到了可以访问 `192.168.100.0/24` 公网上 `OpenStack` 环境的 `Ubuntu` 主机。这台主机将用于运行 `OpenStack` 客户端工具。如果使用的是配套的 `Vagrant` 环境, 可以使用 `controller`

节点，上面安装了提供 neutron 命令行客户端的 python-neutronclient 软件包。

如果使用 Vagrant 创建了该节点，可执行如下命令。

```
vagrant ssh controller
```

确保已经配置好了如下参数（如果用的不是 Vagrant 环境，请修改对应的证书路径和密钥文件路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
vagrant ssh controller
```

操作步骤

本节将创建一个防火墙，具体策略（policy）中的规则（rule）为允许在 TCP 端口 80 上建立连接，并演示具体效果，步骤如下。

1. 首先创建规则：

```
neutron firewall-rule-create \
  --protocol tcp \
  --destination-port 80 \
  --action allow
```

上述命令的输出如图 9-16 所示。

Created a new firewall_rule:	
Field	Value
action	allow
description	
destination_ip_address	
destination_port	80
enabled	True
firewall_policy_id	
id	e43bade8-f50d-4655-bbf3-0030b7aa3dc8
ip_version	4
name	
position	
protocol	tcp
shared	False
source_ip_address	
source_port	
tenant_id	68aa8242ee3c47279fcafa8fa6620311

图 9-16

2. 然后创建一条策略，包含上面创建的规则：

```
neutron firewall-policy-create \
```



```
--firewall-rules "e43bade8-f50d-4655-bbf3-0030b7aa3dc8" \
allow-http-policy
```

上述命令的输出如图 9-17 所示。

Created a new firewall_policy:	
Field	Value
audited	False
description	
firewall_rules	e43bade8-f50d-4655-bbf3-0030b7aa3dc8
id	b12d0be0-9645-448a-b010-a8a8f47580d8
name	allow-http-policy
shared	False
tenant_id	68aa8242ee3c47279fcafa8fa6620311

图 9-17

3. 制定好策略之后，现在可以使用刚创建的策略 ID 创建防火墙：

```
neutron firewall-create
--name cookbook-firewall
b12d0be0-9645-448a-b010-a8a8f47580d8
```

上述命令的输出如图 9-18 所示。

Created a new firewall:	
Field	Value
admin_state_up	True
description	
firewall_policy_id	b12d0be0-9645-448a-b010-a8a8f47580d8
id	1e0c0c8f-f8cf-46e2-bfd2-2b97f39d6ada
name	cookbook-firewall
status	CREATED
tenant_id	68aa8242ee3c47279fcafa8fa6620311

图 9-18

4. 接着可以通过如下命令，列出可用的防火墙：

```
neutron firewall-list
```

上述命令的输出如图 9-19 所示。

id	name	firewall_policy_id
1e0c0c8f-f8cf-46e2-bfd2-2b97f39d6ada	cookbook-firewall	b12d0be0-9645-448a-b010-a8a8f47580d8

图 9-19


5. 还可以通过如下这些命令，查看防火墙、策略、规则的详情。首先来看这条命令：

```
neutron firewall-show 1e0c0c8f-f8cf-46e2-bfd2-2b97f39d6ada
```

上述命令的输出如图 9-20 所示。

Field	Value
admin_state_up	True
description	
firewall_policy_id	b12d0be0-9645-448a-b010-a8a8f47580d8
id	1e0c0c8f-f8cf-46e2-bfd2-2b97f39d6ada
name	cookbook-firewall
status	ACTIVE
tenant_id	68aa8242ee3c47279fcafa8fa6620311

图 9-20

 如果你创建了一个浮动 IP 网络，防火墙的状态只会显示为 **ACTIVE**，因为这时会使用 L3 代理。

接着执行如下命令：

```
neutron firewall-policy-show b12d0be0-9645-448a-b010-a8a8f47580d8
```

上述命令的输出如图 9-21 所示。

Field	Value
audited	False
description	
firewall_rules	e43bade8-f50d-4655-bbf3-0030b7aa3dc8
id	b12d0be0-9645-448a-b010-a8a8f47580d8
name	allow-http-policy
shared	False
tenant_id	68aa8242ee3c47279fcafa8fa6620311

图 9-21

最后执行这条命令：

```
neutron firewall-rule-show e43bade8-f50d-4655-bbf3-0030b7aa3dc8
```

上述命令的输出如图 9-22 所示。

Field	Value
action	allow
description	
destination_ip_address	
destination_port	80
enabled	True
firewall_policy_id	b12d0be0-9645-448a-b010-a8a8f47580d8
id	e43bade8-f50d-4655-bbf3-0030b7aa3dc8
ip_version	4
name	
position	1
protocol	tcp
shared	False
source_ip_address	
source_port	
tenant_id	68aa8242ee3c47279fcafa8fa6620311

图 9-22

6. 现在，在通过 L3 路由器相连的两个不同网络上，启动两个实例。

```
# Create another network
neutron net-create anotherNet
SUBNET_ID=$(neutron subnet-create anotherNet \
    --name anotherSubnet 10.201.0.0/24
    | awk '/\ id\ / {print $4}')

# Connect to cookbook_router_1
neutron router-interface-add cookbook_router_1 ${SUBNET_ID}

# Get Network IDs
NET1=$(neutron net-list | awk '/network_1/ {print $2}')
NET2=$(neutron net-list | awk '/anotherNet/ {print $2}')

# Boot 2 instances
nova boot --flavor 1 --image cirros-image \
    --nic net-id=${NET1} net1-instance
nova boot --flavor 1 --image cirros-image \
    --nic net-id=${NET2} net2-instance
```

7. 在两个实例和网络之间的 L3 路由器上建立起防火墙策略之后，默认会拒绝所有的其他流量。这意味着我们无法建立 ping 或 SSG 连接，即使默认安全组中已经启用这两项特性。我们刚创建的策略规则允许实例之间在 TCP 端口 80 上的流量（来源和目的地均为空，意味着 80 端口在两个方向上都是畅通的）。假设任意一个实例在 80 端口上运行一个 Web 服务器，如 Apache，即可执行如下命令测试防火墙策略规则：

```
wget http://10.200.0.2/
```

如果测试成功，会返回如图 9-23 所示的输出。

```
--2015-02-09 04:22:37-- http://10.200.0.2/
Connecting to 10.200.0.2:80... connected.
HTTP request sent, awaiting response... 200 OK
```

图 9-23



确保本地安全组的规则允许 TCP 端口 80，上面的测试才能成功。

Neutron 的防火墙策略规则不会自动覆盖掉安全组规则。

8. 现在试着使用 SSH 连接到同一个实例。

```
ssh cirros@10.200.0.2
```

ssh 命令会超时，因为防火墙策略目前只允许在 TCP 端口 80 上建立连接。

9. 执行如下步骤，向 Neutron 防火墙添加一条规则。首先创建一个新的防火墙规则，指定为 TCP 端口 22。

```
neutron firewall-rule-create \
    --protocol tcp \
    --destination-port 22 \
    --action allow
```


10. 对于刚创建的规则，我们可以更新现有的策略，或者专门创建一条新策略。本例中，我们新建一条策略：

```
neutron firewall-policy-create \  
  --firewall-rules "451e65e0-alc2-4cc1-8a5d-1ef0608ec1f6" \  
  allow-ssh-policy
```

上述命令的输出如图 9-24 所示。

Created a new firewall_policy:

Field	Value
audited	False
description	
firewall_rules	451e65e0-alc2-4cc1-8a5d-1ef0608ec1f6
id	e78ba041-d1f3-4f9f-93c2-c6f0fc680e86
name	allow-ssh-policy
shared	False
tenant_id	35735709193949ffa8fbfed613ac2d46

图 9-24

11. 然后，更新防火墙，加入这条新策略。

```
neutron firewall-update \  
  --policy e78ba041-d1f3-4f9f-93c2-c6f0fc680e86 \  
  cookbook-firewall
```

你会看到一条信息，表示防火墙已经成功更新。

12. 现在再测试是否能使用 SSH 连接实例。

```
ssh cirros@10.200.0.2
```

现在会成功连接（不过可能会因为证书错误而失败，实际上我们看到的是验证失败的消息，但是这表明 TCP 端口 22 是畅通的）。

工作原理

Neutron 防火墙是策略驱动的防火墙规则，设置在于不同子网间的路由器上。这允许 OpenStack 环境的网络管理员创建网络级别的策略，而不是依靠用户来管理安全组规则。

创建 Neutron 防火墙，需要执行如下步骤：

- 1. 通过 neturon firewall-rule-create 命令，创建防火墙规则；
- 2. 通过 neutron firewall-policy-create \$RULE_ID 命令，创建策略；
- 3. 通过 neutron firewall-create \$POLICY_ID 命令，创建带策略的防火墙。

在使用 Neutron 防火墙时，规则的优先级顺序如下：

- 1. Neutron 防火墙策略；

2. 安全组规则;
3. 实例内部基于主机的规则 (如 iptables)。

一定要确保实例上运行的服务被正确访问; 安全组规则正是为了访问这些服务而设置的。换句话说, 网络管理员可能设置了策略, 允许 TCP 端口 80 通过子网间的 L3 路由, 但是还需要在实例上应用一个安全组规则来允许 TCP 端口 80 上的连接。

9.10 安装 OpenStack 编排服务 Heat

Heat 是 OpenStack 编排服务, 提供了一个基于模板的系统来定义 OpenStack 中的环境和资源。据了解, 只需要使用 OpenStack Heat 以及一些最基础的资源, 即可部署 OpenStack。Heat 可以用来描述计算资源, 安装软件, 以及定义负载均衡与数据库的关系。

准备工作

确保有一台运行 OpenStack 组件的服务器。如果你使用的是配套 Vagrant 环境, 可使用前言中提到的控制节点。

确保登录到环境中的控制节点。如果通过 Vagrant 创建了这两个节点, 可执行如下命令。

```
vagrant ssh controller
```

操作步骤

在控制节点上执行如下步骤, 安装 Heat 服务。

1. 使用如下命令, 创建名为 heat 的数据库。

```
MYSQL_ROOT_PASS=openstack
```

```
mysql -uroot
      -p$MYSQL_ROOT_PASS
      -e 'CREATE DATABASE heat;'
```

2. 创建一个名为 heat 的用户, 密码设为 openstack, 赋予使用该数据库的权限。

```
MYSQL_HEAT_PASS=openstack
```

```
mysql -uroot -p${MYSQL_ROOT_PASS}
      -e "GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'%'
      IDENTIFIED BY '${MYSQL_HEAT_PASS}';"
```

```
mysql -uroot -p${MYSQL_ROOT_PASS}
      -e "GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'localhost'
      IDENTIFIED BY '${MYSQL_HEAT_PASS}';"
```

3. Keystone 服务需要知道启用了 Heat, 因此首先确保有针对 Heat 服务的 Keystone 证书, 执行如下命令即可。

```
keystone user-create \
  --name=heat \
  --pass=heat \
  --email=heat@localhost
```

```
keystone user-role-add \
  --user=heat \
  --tenant=service \
  --role=admin
```

4. 然后, 执行如下命令, 在 Keystone 中为 Heat 添加两个服务和端点。

```
keystone service-create \
  --name=heat \
  --type=orchestration \
  --description="Heat Orchestration API"
```

```
ORCHESTRATION_SERVICE_ID=$(keystone service-list \
  | awk '/\ orchestration\ / {print $2}')
```

```
keystone endpoint-create
  --region RegionOne \
  --service-id=${ORCHESTRATION_SERVICE_ID} \
  --publicurl=http://172.16.0.200:8004/v1/${tenant_id}s \
  --internalurl=http://172.16.0.200:8004/v1/${tenant_id}s \
  --adminurl=http://172.16.0.200:8004/v1/${tenant_id}s
```

```
keystone service-create \
  --name=heat-cfn \
  --type=cloudformation \
  --description="Heat CloudFormation API"
```

```
CLOUDFORMATION_SERVICE_ID=$(keystone service-list \
  | awk '/\ cloudformation\ / {print $2}')
```

```
keystone --insecure endpoint-create \
  --region RegionOne \
  --service-id=${CLOUDFORMATION_SERVICE_ID} \
  --publicurl=http://172.16.0.200:8000/v1/ \
  --internalurl=http://172.16.0.200:8000/v1 \
  --adminurl=http://172.16.0.200:8000/v1
```

5. 现在可使用 apt 安装 Heat 服务所需要的软件包。

```
sudo apt-get install heat-api heat-api-cfn heat-engine
```

6. Heat 的配置文件时/etc/heat/heat.conf。编辑该文件, 加入如下设置。

```
[DEFAULT]
rabbit_host=172.16.0.200
rabbit_port=5672
rabbit_userid=guest
rabbit_password=guest
rabbit_virtual_host=/
rabbit_ha_queues=false
log_dir=/var/log/heat
```



```

[database]
backend=sqlalchemy
connection = mysql://heat:openstack@172.16.0.200/heat

[keystone_authtoken]
auth_uri = https://192.168.100.200:35357/v2.0
identity_uri = https://192.168.100.200:5000
admin_tenant_name = service
admin_user = heat
admin_password = heat
insecure = True
heat_watch_server_url = http://192.168.100.200:8003
heat_waitcondition_server_url = http://192.168.100.200:8000/v1/waitcondition
heat_metadata_server_url = http://192.168.100.200:8000

[clients]
endpoint_type = internalURL

[clients_ceilometer]
endpoint_type = internalURL

[clients_cinder]
endpoint_type = internalURL

[clients_heat]
endpoint_type = internalURL

[clients_keystone]
endpoint_type = internalURL

[clients_neutron]
endpoint_type = internalURL

[clients_nova]
endpoint_type = internalURL

[clients_swift]
endpoint_type = internalURL

[clients_trove]
endpoint_type = internalURL

[ec2authtoken]
auth_uri = https://192.168.100.200:5000/v2.0

[heat_api]
bind_port = 8004

[heat_api_cfn]
bind_port = 8000

[heat_api_cloudwatch]
bind_port = 8003

```

7. 在启动服务之前，必须使用如下命令，创建 Heat 数据库表以初始项。

```
heat-manage db_sync
```

8. 最后，通过如下命令，重启服务。

```
service heat-api restart
service heat-api-cfn restart
service heat-engine restart
```

工作原理

与其他 OpenStack 服务一样，Heat 要求数据库后端中存有 Keystone 中定义的授权证书和服务端点。我们随后在 `/etc/heat/heat.conf` 文件中，使用这些证书来描述该服务。

值得注意的是，要在启动服务器之前，通过 `heat-manage db_sync` 命令初始化数据库。这会准备好数据库中的表结构，方便 Heat 服务使用。

9.11 使用 Heat 启动实例

使用 Heat，可以创建各种各样的模板，从启动简单的实例到创建完整的应用环境都可完成。本节将讲解 Heat 的基础知识，通过 Heat 启动一个实例，将其加入现有的 Neutron 网络，并指定一个浮动 IP。Heat 模板被称为 Heat 编排模板（Heat Orchestration Templates, HOT），是基于 YAML 语言的文件。它们描述了所使用的资源、实例的类型和大小、实例所连接的网路，以及其他运行该环境所需的信息。

本节中，我们将介绍如何使用 HOT 文件，启动两台运行 Apache 的 Web 服务器，并连接到一个运行 HA Proxy 的实例。

准备工作

确保登录到了可以访问 192.168.100.0/24 公网上 OpenStack 环境的 Ubuntu 主机。这台主机将用于运行 OpenStack 客户端工具。如果使用的是配套的 Vagrant 环境，可以使用 controller 节点，上面安装了提供 heat 命令行客户端的 `python-heatclient` 软件包。



如果使用的是配套 Vagrant 环境，公共网络并不会像物理环境一样连接到因特网。实例将无法访问、安装 Heat 模板中描述的远程软件包。为了解决这个问题，可以在运行虚拟环境的主机上启动一个代理服务器，如 Squid。然后配置代理服务器允许访问 192.168.100.0/24 网络。示例 HOT 文件默认 Squid 运行在 `http://192.168.100.1:3128` 上，这是在物理主机上分配的一个地址。

如果使用 Vagrant 创建了该节点，可执行如下命令。

```
vagrant ssh controller
```

确保已经配置好了如下参数（如果用的不是 Vagrant 环境，请修改对应的证书路径和密钥文件路径）。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.200:5000/v2.0/
export OS_NO_CACHE=1
export OS_KEY=/vagrant/cakey.pem
export OS_CACERT=/vagrant/ca.pem
```

操作步骤

在本节中，我们将下载一个名为 `cookbook.yaml` 的 HOT 文件，该文件描述了实例的具体情况以及要附加的网络，具体步骤如下。

1. 首先，使用如下命令，从本书的 Github 仓库中下载 HOT 文件。

```
wget -O cookbook.yaml \
https://raw.githubusercontent.com/OpenStackCookbook/
OpenStackCookbook/master/cookbook.yaml
```

2. Heat 接受来自命令行的输入参数，以及传入模板中的环境文件。这些参数会出现在 HOT 文件的顶部，如下所示。

```
parameters:
  key_name:
    type: string
    description: Name of keypair to assign to servers
  image:
    type: string
    description: Name of image to use for servers
  flavor:
    type: string
    description: Flavor to use for servers
  public_net_id:
    type: string
    description: >
      ID of public network for which floating IP addresses will be allocated
  private_net_id:
    type: string
    description: ID of private network into which servers get deployed
  private_subnet_id:
    type: string
    description: ID of private sub network into which servers get deployed
```

3. 在启动该模板文件时，必须传入各种参数。执行如下命令，确保参数设置正确。

```
nova keypair-list
nova image-list
```



```
nova flavor-list
neutron net-list
```

neutron-netlist 命令的输出如图 9-25 所示。

id	name	subnets
5e5d24bd-9d1f-4ed1-84b5-0b7e2a9a233b	ext_net	11c11dca-479b-435d-889d-fc29479b0a24 192.168.100.0/24
25153759-994f-4835-9b13-bf0ec77fb336	cookbook_network_1	4cf2c09c-b3d5-40ed-9127-ec40e5e38343 10.200.0.0/24

图 9-25

4. 得到详细信息之后，创建一个环境文件，用于存储参数。在启动栈时，这些参数将传入 HOT 文件。在 `cookbook.yaml` 文件所在目录下，创建 `cookbook-env.yaml` 文件，根据上面命令的输出写入如下行。

```
parameters:
  key_name: demokey
  image: trusty-image
  flavor: ml.tiny
  public_net_id: 5e5d24bd-9d1f-4ed1-84b5-0b7e2a9a233b
  private_net_id: 25153759-994f-4835-9b13-bf0ec77fb336
  private_subnet_id: 4cf2c09c-b3d5-40ed-9127-ec40e5e38343
```

5. 现在可通过如下命令启动栈了：

```
heat stack-create haproxy101 \
  --template-file=cookbook.yaml \
  --environment-file=cookbook-env.yaml
```

上述命令的输出如图 9-26 所示。

id	stack_name	stack_status	creation_time
5484d8ec-62ae-4c59-80f2-ce9a8ada9098	haproxy101	CREATE_IN_PROGRESS	2015-02-23T16:38:54Z

图 9-26

6. 执行如下命令，查看栈列表：

```
heat stack-list
```

之后会看到当前正在运行的栈列表，如图 9-27 所示。

id	stack_name	stack_status	creation_time
5484d8ec-62ae-4c59-80f2-ce9a8ada9098	haproxy101	CREATE_COMPLETE	2015-02-23T16:38:54Z

图 9-27

7. 模板中有一个部分设置了如何查看输出。输出可让用户查看栈的情况；如果没有这些信息，用户必须在系统中进行细致的摸索，才能找到组成栈的实例所赋予的 IP 地址。执行如下命令，查看正在运行的实例的相关输出：

heat output-list

上述命令的输出如图 9-28 所示。

output_key	description
haproxy_public_ip	Floating IP address of haproxy in public network
webserver1_private_ip	IP address of webserver1 in private network
webserver2_private_ip	IP address of webserver2 in private network

图 9-28

8. 如需查看某个特定的值, 可访问 HA Proxy 的公共 IP (即浮动 IP), 这样就能访问运行在负载均衡的私有地址上的网站。

heat output-show haproxy101 haproxy_public_ip

上述命令会显示 HA Proxy 的 IP 地址。

192.168.100.12

9. 然后使用 <http://192.168.100.12> 地址, 向 HA Proxy 负载均衡后的 Web 服务器发送请求。

工作原理

HOT 是描述环境的 YAML 文件 (也称为栈), 模板的大致结构如下。

```
description:
parameters:
resources:
outputs:
```

- ❑ **description:**部分: 告诉用户使用该模板时会发生什么。
- ❑ **parameters:**部分: 定义输入变量, 如使用的镜像类型、实例所在的网络、实例相关联的密钥对名称。参数是任意的, 可包含成功执行模板所需的任何信息。
parameters:部分也可通过环境文件导入 (通过 `--environment-file=parameter` 指定)。每个参数必须有一个默认值, 或在环境文件中指定, 栈才能成功启动。
- ❑ **resources:**部分: 这部分内容最大, 因为它是对环境的具体描述, 包含使用的实例, 如何命名, 处在哪个网络, 各个元素之间的关系, 环境如何编排等。如何写好这部分内容超出了本书的范围。
- ❑ **outputs:**部分: 指的是栈执行后的返回值。例如, 用户需要知道如何访问刚刚创建的栈。栈运行时可能会被赋予随机 IP 和主机名, 因此要能够掌握正确的信息, 才能访问该环境。

第 10 章

使用 OpenStack Dashboard

本章将讲述以下内容：

- ☐ 安装 OpenStack Dashboard
- ☐ 使用 OpenStack Dashboard 进行密钥管理
- ☐ 使用 OpenStack Dashboard 管理 Neutron 网络
- ☐ 使用 OpenStack Dashboard 进行安全组管理
- ☐ 使用 OpenStack Dashboard 启动实例
- ☐ 使用 OpenStack Dashboard 终止实例
- ☐ 使用 OpenStack Dashboard 连接到使用 VNC 的实例
- ☐ 使用 OpenStack Dashboard 添加新租户项目
- ☐ 使用 OpenStack Dashboard 进行用户管理
- ☐ 使用 OpenStack Dashboard 进行 LBaaS 操作
- ☐ 使用 OpenStack Dashboard 进行 OpenStack 编排

10.1 简介

通过命令行接口来管理 OpenStack 环境可以完全控制这个云环境，但是通过使用 Web 界面，运营人员和管理员管理云环境和实例会更轻松一些。OpenStack Dashboard（仪表盘），即 Horizon，提供了基于 Web 的 GUI。它是一个可以运行在 Apache 服务器上的 Web 服务，使用 Python 的 **Web Service Gateway Interface**（WSGI）和 Django（一个 Web 快

速开发框架)。

安装好 OpenStack Dashboard 之后, 我们就可以管理 OpenStack 环境中的所有组件。

10.2 安装 OpenStack Dashboard

使用 Ubuntu 的软件包资源库安装 OpenStack Dashboard 的过程非常简单直接。

准备工作

确保已经登录到 OpenStack 控制节点。如果这一节点是像 1.2 节中描述的那样使用 Vagrant 创建的, 则可以使用如下命令来访问。

```
vagrant ssh controller
```

操作步骤

要安装 OpenStack Dashboard, 需要执行以下步骤安装所需的包和依赖。

1. 安装所需的包, 如下所示。

```
sudo apt-get update
sudo apt-get install openstack-dashboard
```

2. 可以通过编辑 `/etc/openstack-dashboard/local_setting.py` 文件, 来配置 OpenStack Dashboard。

```
OPENSTACK_HOST = "192.168.100.200"
OPENSTACK_KEYSTONE_URL = "http://%s:5000/v2.0" %
OPENSTACK_HOST
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "_member_"
CACHES = {
    'default': {
        'BACKEND':
        'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
ALLOWED_HOSTS = ''
```

3. 现在需要配置 OpenStack Dashboard 来使用 VNC 代理服务, 该服务可以通过 OpenStack Dashboard 接口调用。为此, 将以下内容添加到 `/etc/nova/nova.conf` 文件中。

```
# NoVNC
novnc_enabled=true
novncproxy_host=192.168.100.200
novncproxy_base_url=http://192.168.100.200:6080/vnc_auto.html
novncproxy_port=6080
```

```
xvpvncproxy_port=6081
xvpvncproxy_host=192.168.100.200
xvpvncproxy_base_url=http://192.168.100.200:6081/console
```

```
vncserver_proxycient_address=192.168.100.200
vncserver_listen=0.0.0.0
```

4. 重启 nova-api 服务, 执行以上变更。

```
sudo restart nova-api
sudo restart nova-compute
sudo service apache2 restart
```



Ubuntu 中的 OpenStack Dashboard 和标准的 Dashboard 风格稍微有些不太一样, 但基本功能都是一样的。Ubuntu 仅仅添加了一些额外的特性, 让用户可以下载 Canonicals 中的编排工具 Juju。可以使用如下命令移除这个特性:

```
sudo dpkg --purge openstack-dashboard-ubuntu-theme
```

工作原理

使用 Ubuntu 软件包资源库即可安装 OpenStack Dashboard(即 Horizon)。由于 OpenStack Dashboard 运行在 Apache 服务器上, 需要重启服务器使得变更生效。

还要加入 VNC 代理服务, 它支持通过 Web 接口从网络中访问实例。

本章所示的截图均为删除 Ubuntu 自定义主题后的界面。

10.3 使用 OpenStack Dashboard 进行密钥管理

SSH 密钥对 (key pair) 允许用户无需输入密码就能连接到 Linux 实例, 几乎所有 OpenStack 中的 Linux 镜像都将它作为默认访问机制。用户可以通过 OpenStack Dashboard 来管理自己的密钥对。通常, 这也是一个新用户进入 OpenStack 环境后需要完成的第一个任务。

准备工作

打开浏览器, 指向 OpenStack Dashboard 地址 <http://192.168.100.200/>, 并以 demo 用户身份 (1.6 节创建的) 登录, 密码为 openstack。

操作步骤

登录用户的密钥对的管理的具体步骤将在下面几个小节中讨论。

添加密钥对

密钥对可以通过以下步骤添加。

1. 点击 **Project | Compute** 菜单下的 **Access & Security** 选项卡，可以为系统添加一个新的密钥对，如图 10-1 所示。

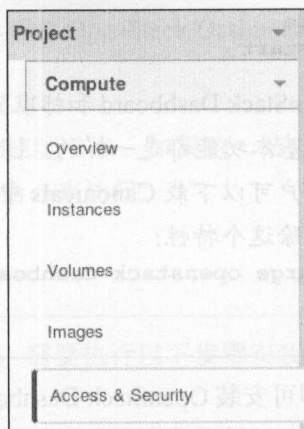


图 10-1

2. 现在可以看到允许访问安全设置和密钥对管理的页面。在 **Key Pairs** 选项卡下有一个有效密钥对列表，启动和方位实例时可以使用。点击 **Create Key Pair** 按钮，创建一个新的密钥对，如图 10-2 所示。

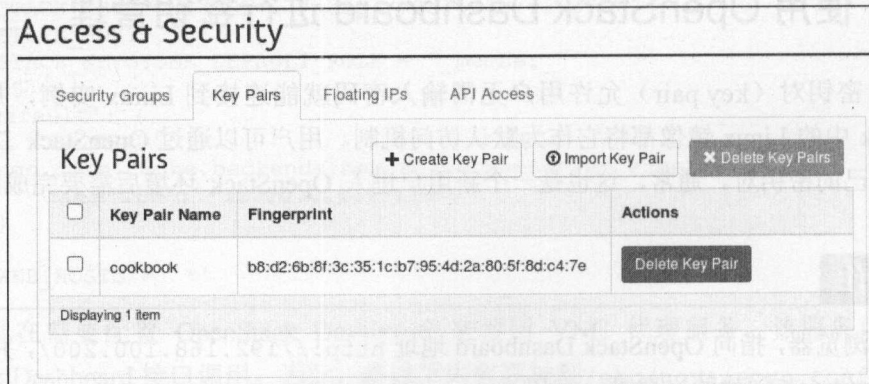
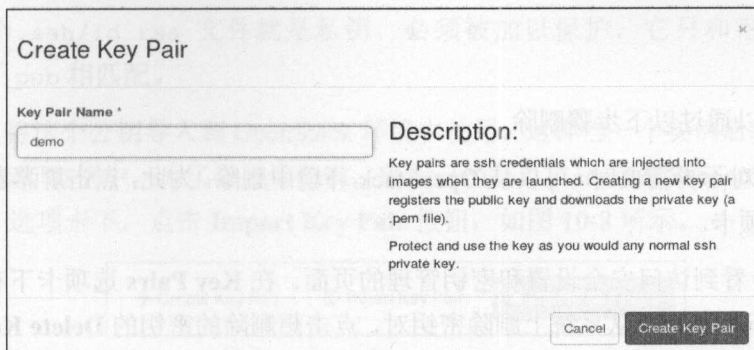


图 10-2

3. 在 **Create Key Pair** 界面输入名称（如 demo），确保名称中没有空格，然后点击 **Create Key Pair** 按钮，如图 10-3 所示。



Create Key Pair

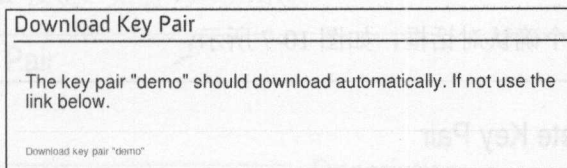
Key Pair Name *
demo

Description:
Key pairs are ssh credentials which are injected into images when they are launched. Creating a new key pair registers the public key and downloads the private key (a .pem file).
Protect and use the key as you would any normal ssh private key.

Cancel Create Key Pair

图 10-3

4. 一旦密钥创建成功，系统就会提示用户是否保存密钥对中的私有密钥到硬盘，如图 10-4 所示。




Download Key Pair

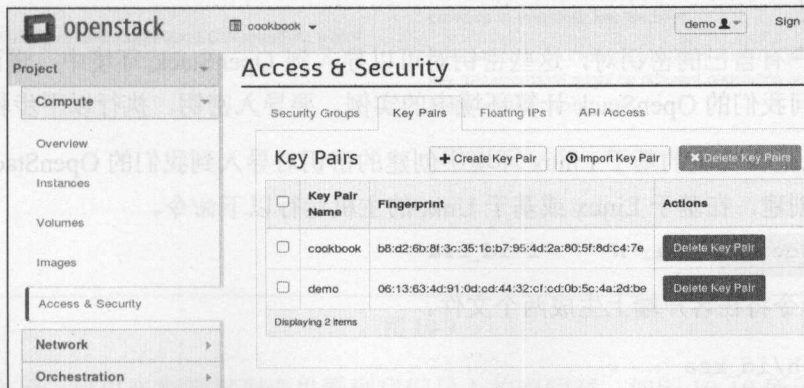
The key pair "demo" should download automatically. If not use the link below.

[Download key pair "demo"](#)

图 10-4

 私有 SSH 密钥不能重复创建，所以应在文件系统上安全地保存。

5. 点击 **Access & Security** 选项卡返回密钥对列表。现在就能看到新创建的密钥对。启动实例时，可以选择新密钥对，并通过保存在本地的私有密钥访问该实例，如图 10-5 所示。



openstack cookbook demo Sign

Access & Security

Security Groups Key Pairs Floating IPs API Access

Key Pairs + Create Key Pair Import Key Pair Delete Key Pairs

Key Pair Name	Fingerprint	Actions
<input type="checkbox"/> cookbook	b8:d2:6b:8f:3c:35:1c:b7:95:4d:2a:60:5f:6d:c4:7e	Delete Key Pair
<input type="checkbox"/> demo	06:13:63:4d:91:0d:cd:44:32:cf:cd:0b:5c:4a:2d:be	Delete Key Pair

Displaying 2 items

Project: Compute

Overview
Instances
Volumes
Images
Access & Security
Network
Orchestration

图 10-5

删除密钥对

密钥对可以通过以下步骤删除。

1. 当密钥对不再需要时, 可以从 OpenStack 环境中删除。为此, 点击屏幕左侧的 **Access & Security** 选项卡。

2. 然后会看到访问安全设置和密钥管理的页面。在 **Key Pairs** 选项卡下有一个密钥对列表, 用来访问实例。要从系统上删除密钥对, 点击想删除的密钥的 **Delete Key Pair** 按钮, 如图 10-6 所示。

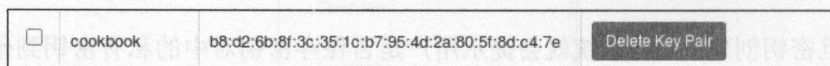


图 10-6

3. 接着会看到一个确认对话框, 如图 10-7 所示。

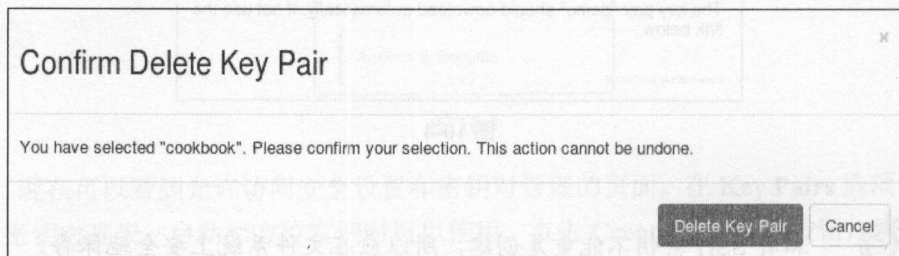


图 10-7

点击 **Delete Key Pair** 按钮后, 该密钥对就会被删除。

导入密钥对

如果用户有自己的密钥对, 这些密钥对可以导入到 OpenStack 环境中, 进而可以使用它们继续访问我们的 OpenStack 计算环境中的实例。要导入密钥, 执行以下步骤。

1. 可以把在传统的基于 Linux 环境中创建的密钥对导入到我们的 OpenStack 设置中。如果还没有创建, 在基于 Linux 或基于 Unix 的主机运行以下命令。

```
ssh-keygen -t rsa -N "" -f id_rsa
```

2. 该命令将在客户端上生成两个文件。

☐ .ssh/id_rsa

☐ .ssh/id_rsa.pub

3. 这个 `.ssh/id_rsa` 文件就是私钥，必须被加以保护，它只和密钥对中的公钥 `ssh/id_rsa.pub` 相匹配。

4. 可以将这个公钥导入到 OpenStack 环境中使用。这样当一个实例启动时，公钥就会被注入到该运行实例中。为了导入该公钥，确保已经打开 **Access & Security** 界面，然后进入 **Key pairs** 选项卡下，点击 **Import Key Pair** 按钮，如图 10-8 所示。

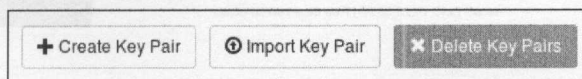


图 10-8

5. 然后会看到一个提示页面，询问密钥对的名称，把公钥的内容复制过来。为该公钥命名，然后复制粘贴公钥的内容（如 `.ssh/id_rsa.pub` 的内容）到空白处。输入之后，点击 **Import Key Pair** 按钮，如图 10-9 所示。

Import Key Pair

Key Pair Name *

Public Key *

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQCIINuz
Yld+Q2sIJk7+Pp34yvyDO5/lmf1wEGX9Q
/9F64pHvnV3E3lhphr2zSrV1L1DxeIKUzlxOCimhod
3D7EkWDA3X8o/ZTcz7dTgV+6y
/maXE3GXrtVj1Fwn
/5WaWksQumVYcB6Nwu0i7kLE4BZ5nCLJHr7MwI8
RnkB1IH6aMZa3z4bGBKINpb8poUFSckXerjvzpz
GfvKcal3pCEyGC0SqID66GIISETXsjnGbxvV9JO3
V
/inuZshJk25qe9HGB274lpmECI8QwYafNKLVIHj07
```

Description:

Key Pairs are how you login to your instance after it is launched.

Choose a key pair name you will recognise and paste your SSH public key into the space provided.

SSH key pairs can be generated with the `ssh-keygen` command:

```
ssh-keygen -t rsa -f cloud.key
```

This generates a pair of keys: a key you keep private (`cloud.key`) and a public key (`cloud.key.pub`). Paste the contents of the public key file here.

After launching an instance, you login using the private key (the username might be different depending on the image you launched):

```
ssh -i cloud.key <username>@<instance_ip>
```

图 10-9

完成之后，可以在密钥对列表里看到我们导入的密钥对，如图 10-10 所示。

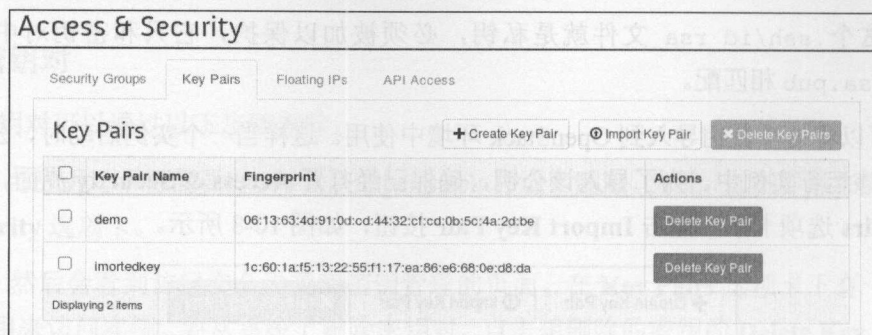


图 10-10

工作原理

密钥对管理非常重要，提供了一个一致性的、安全的方法访问运行实例。允许用户在租户中创建、删除和导入密钥对，维护系统安全。

OpenStack Dashboard 允许用户通过简单的方式创建密钥对。但用户必须保证下载的私钥的安全。

删除密钥对也非常简单，但是该用户必须清楚是否还有运行的实例在使用该密钥对，否则该用户将不能再访问这些系统。因为每个创建的密钥对都是独一无二的，即使为密钥起了相同的名字。

导入密钥对的好处是用户可以复用已有的安全密钥对，这些密钥对在 OpenStack 环境之外一直被使用，并且在新的私有云环境中还可以继续使用。这为用户从一个环境迁移到另一个环境提供了一致的用户体验。

10.4 使用 OpenStack Dashboard 管理 Neutron 网络

使用 OpenStack Dashboard 可以查看、创建和编辑 Neutron 网络，从而让管理复杂的软件定义的网络变得更简单。某些特定的功能需要用户以 admin 权限登录 OpenStack Dashboard，例如创建共享网络和路由器。不过，任何用户都是有权限创建私有网络的。OpenStack Dashboard 会自动刷新网络的拓扑结构，这能帮助用户管理复杂的软件定义的网络。

准备工作

打开浏览器，指向 OpenStack Dashboard 地址 <http://192.168.100.200>，并以 demo 用户身份（1.6 节创建的）登录，密码为 openstack。

操作步骤

本节中，我们将学习以下话题：

- ☐ 创建网络
- ☐ 删除网络
- ☐ 查看网络

创建网络

要以登录用户身份创建一个私有网络，需要执行以下步骤。

1. 选中如图 10-11 所示截图中的 **Network** 选项卡。

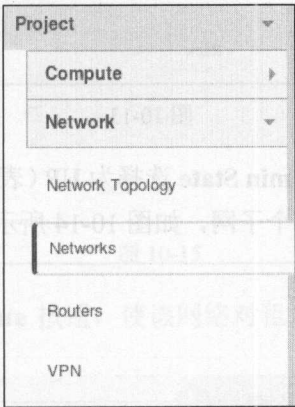


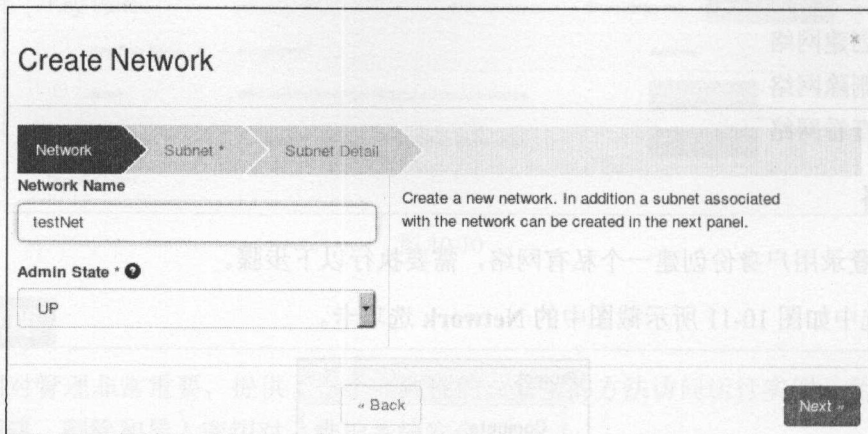
图 10-11

2. 选中后，页面中会列出所有可以分配给实例的网络，如图 10-12 所示。

Networks						
Networks						
				+ Create Network		✕ Delete Networks
<input type="checkbox"/>	Name	Subnets Associated	Shared	Status	Admin State	Actions
<input type="checkbox"/>	ext_net	cookbook_float_subnet_1 192.168.100.0/24	No	ACTIVE	UP	Edit Network ▾
<input type="checkbox"/>	cookbook_network_1	cookbook_subnet_1 10.200.0.0/24	No	ACTIVE	UP	Edit Network ▾
Displaying 2 items						

图 10-12

3. 点击 **Create Network** 按钮，创建一个新的网络。
4. 此时会弹出一个对话框要求输入网络的名字，如图 10-13 所示。



Create Network

Network Subnet * Subnet Detail

Network Name

testNet

Admin State *

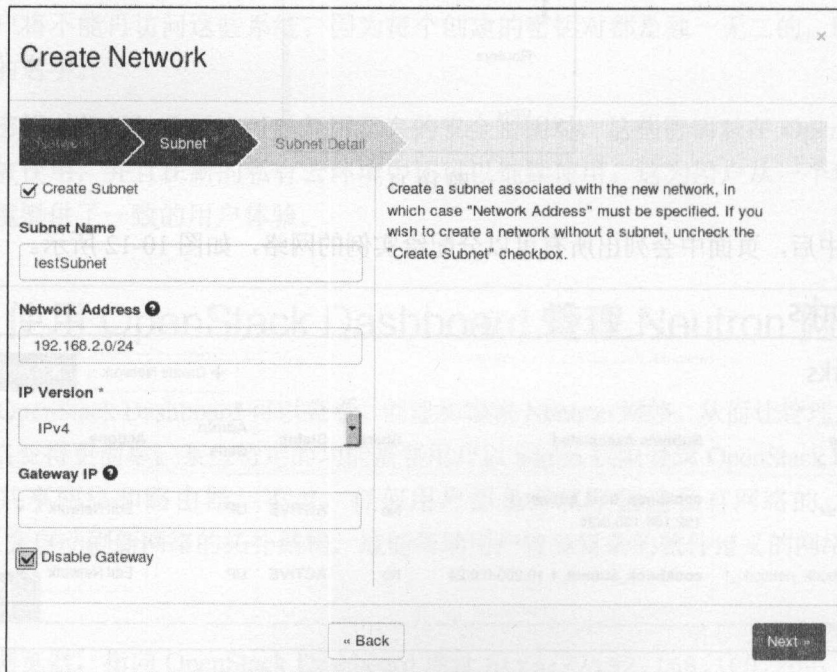
UP

Create a new network. In addition a subnet associated with the network can be created in the next panel.

« Back Next »

图 10-13

5. 选择一个名字，并确认 **Admin State** 选择为 **UP**（表示启用该网络并允许实例接入）。接着选择 **Subnet** 选项卡来分配一个子网，如图 10-14 所示。



Create Network

Network Subnet Subnet Detail

☒ Create Subnet

Subnet Name

testSubnet

Network Address *

192.168.2.0/24

IP Version *

IPv4

Gateway IP *

☒ Disable Gateway

Create a subnet associated with the new network, in which case "Network Address" must be specified. If you wish to create a network without a subnet, uncheck the "Create Subnet" checkbox.

« Back Next »

图 10-14

6. 填写完子网信息后, 选择 **Subnet Detail** 选项卡, 配置诸如 DHCP 范围、DNS、备用路由等用户所需的更多细节, 如图 10-15 所示。

Create Network

Network Subnet Detail

☒ Enable DHCP Specify additional attributes for the subnet.

Allocation Pools

DNS Name Servers

8.8.8.8
8.8.4.4

Host Routes

0.0.0.0/0, 192.168.2.254

Back Create

图 10-15

7. 输入完成后, 点击 **Create** 按钮, 使该网络对租户中的用户生效, 并返回到可用网络列表, 如图 10-16 所示。

Networks						
Networks						
				+ Create Network		✖ Delete Networks
<input type="checkbox"/>	Name	Subnets Associated	Shared	Status	Admin State	Actions
<input type="checkbox"/>	testNet	testSubnet 192.168.2.0/24	No	ACTIVE	UP	Edit Network ▼
<input type="checkbox"/>	ext_net	cookbook_float_subnet_1 192.168.100.0/24	No	ACTIVE	UP	Edit Network ▼
<input type="checkbox"/>	cookbook_network_1	cookbook_subnet_1 10.200.0.0/24	No	ACTIVE	UP	Edit Network ▼
Displaying 3 items						

图 10-16

删除网络

要以登录用户身份删除一个私有网络, 执行以下步骤。

1. 选中如图 10-17 所示截图中的 **Network** 选项卡。

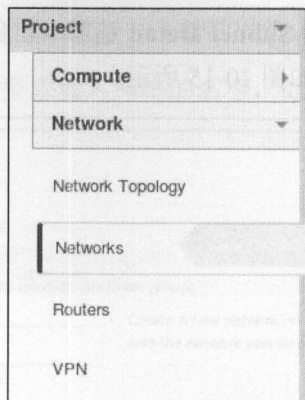


图 10-17

2. 选中后，页面中会列出所有可以分配给实例的网络，如图 10-18 所示。

Networks						
Networks						
			+ Create Network		X Delete Networks	
<input type="checkbox"/>	Name	Subnets Associated	Shared	Status	Admin State	Actions
<input type="checkbox"/>	testNet	testSubnet 192.168.2.0/24	No	ACTIVE	UP	Edit Network ▼
<input type="checkbox"/>	ext_net	cookbook_float_subnet_1 192.168.100.0/24	No	ACTIVE	UP	Edit Network ▼
<input type="checkbox"/>	cookbook_network_1	cookbook_subnet_1 10.200.0.0/24	No	ACTIVE	UP	Edit Network ▼
Displaying 3 items						

图 10-18

3. 要删除一个网络，选中网络名字后面的复选框并点击 **Delete Networks** 按钮，如图 10-19 所示。

Networks						
Networks						
			+ Create Network		X Delete Networks	
<input type="checkbox"/>	Name	Subnets Associated	Shared	Status	Admin State	Actions
<input checked="" type="checkbox"/>	testNet	testSubnet 192.168.2.0/24	No	ACTIVE	UP	Edit Network ▼
<input type="checkbox"/>	ext_net	cookbook_float_subnet_1 192.168.100.0/24	No	ACTIVE	UP	Edit Network ▼
<input type="checkbox"/>	cookbook_network_1	cookbook_subnet_1 10.200.0.0/24	No	ACTIVE	UP	Edit Network ▼
Displaying 3 items						

图 10-19

4. 此时会弹出一个对话框要求确认删除操作，如图 10-20 所示。

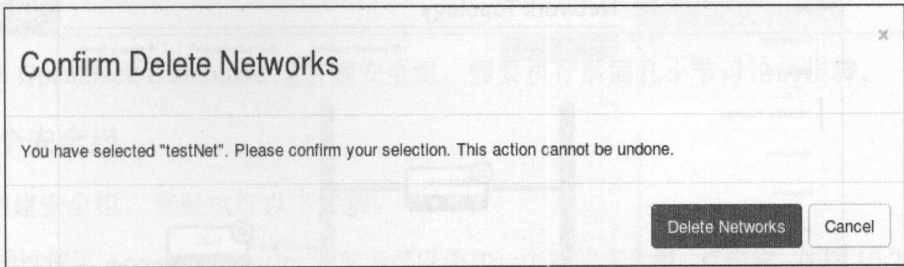



图 10-20

5. 点击 **Delete Networks** 按钮后，网络将被删除，然后返回到可用网络列表。

 一个网络只有当没有任何实例接入它里面的情况下才能被删除，否则会提示不允许删除该网络因为仍有实例连接在上面。

查看网络

OpenStack Dashboard 可以让用户和管理员查看网络环境的拓扑结构。执行如下步骤来查看网络拓扑。

1. 要在 OpenStack Dashboard 中管理网络，首先应选中图 10-21 所示的 **Network** 选项卡。

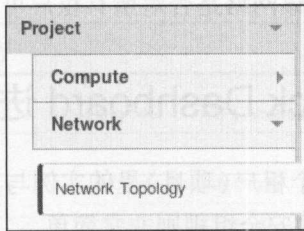


图 10-21

2. 点击 **Network Topology** 选项卡，此时会打开一个很不错的界面展示所有网络的概况以及连接在上面的实例，如图 10-22 所示。

3. 该界面中每个部分都是可以点击的，如网络（点击进入网络管理界面）、实例（点击进入实例管理界面），并可以在界面中创建网络、路由和启动新实例。

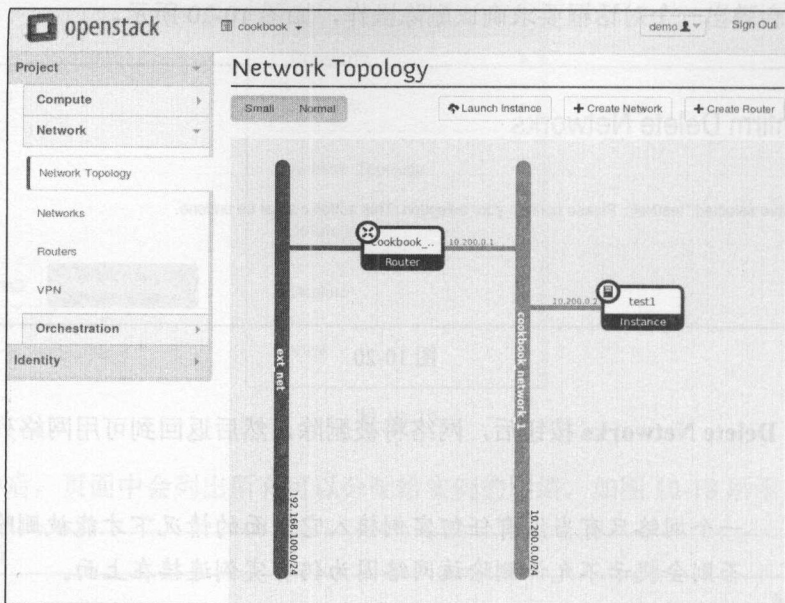


图 10-22

工作原理

管理和编辑 Neutron 网络是 OpenStack Grizzly 版中的新特性。Neutron 网络管理起来可能非常复杂，而 OpenStack Dashboard 这种可视化界面能让网络管理工作变得容易得多。

管理员（拥有 admin 角色的用户）可以创建共享网络。步骤与上述步骤相同，只是多出来一个额外的选项，可以选择该网络是否对所有租户可见。

10.5 使用 OpenStack Dashboard 进行安全组管理

安全组是网络规则，允许一个租户（项目）里的实例与其他实例隔离开。使用 OpenStack Dashboard 管理 OpenStack 实例的安全组规则非常简单。



正如第 1 章所述，项目（project）和租户（tenant）实际上指的是同一个东西，所以在 OpenStack Dashboard 中，租户指的就是项目，而 Keystone 项目指的就是租户。

准备工作

打开浏览器，指向 OpenStack Dashboard 地址 <http://192.168.100.200>，并以 demo

用户身份（1.6 节创建的）登录，密码为 openstack。

操作步骤

要在 OpenStack Dashboard 里管理安全组，需要执行后面几小节讨论的步骤。

创建一个安全组

要创建安全组，需要执行以下步骤。

- 1. 通过使用 **Access & Security** 选项卡可以添加一个新的安全组，点击它，如图 10-23 所示。

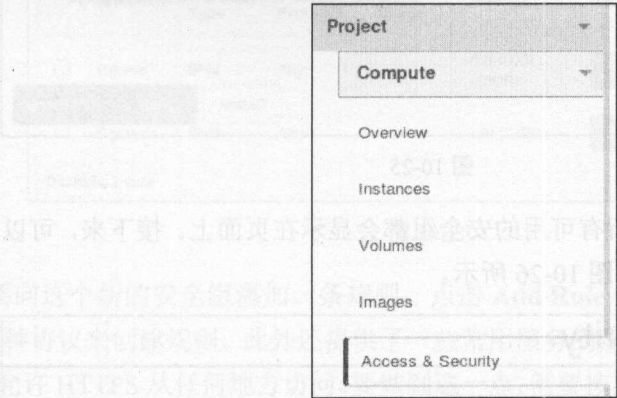


图 10-23

- 2. 这时会看到访问安全设置和密钥管理的页面。在 **Security Groups** 选项卡下有一个安全组列表，可供实例启动时使用。点击 **Create Security Group** 按钮，创建一个新的密钥对，如图 10-24 所示。

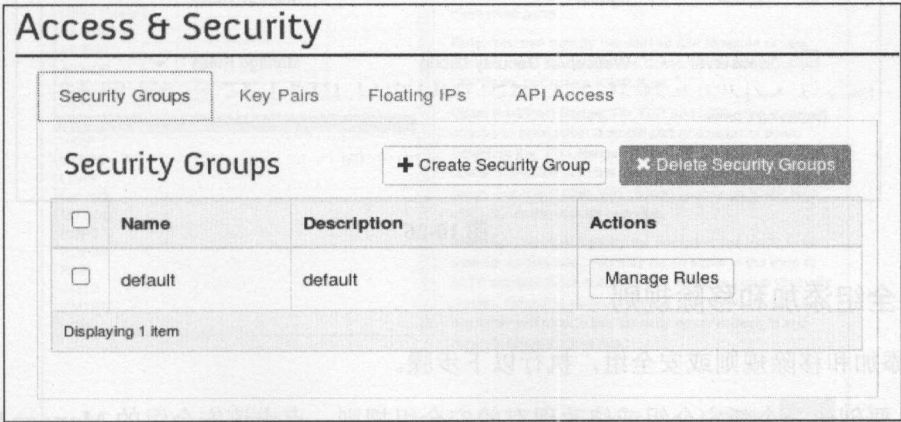
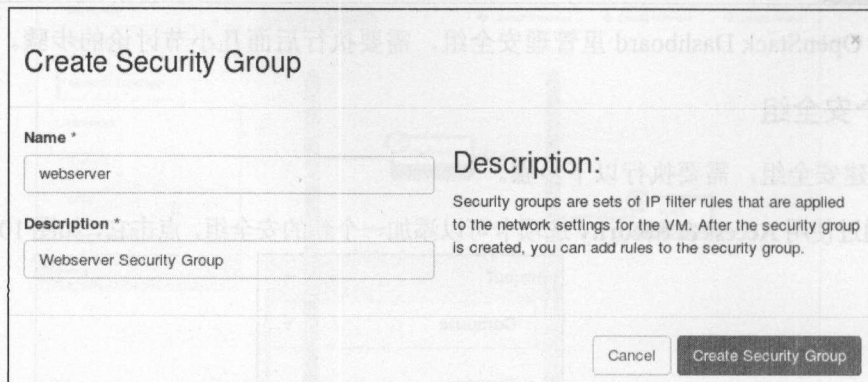


图 10-24

接着会看到创建页面，输入安全组名称（不能有空格）并提供一段描述，如图 10-25 所示。



The image shows a 'Create Security Group' form. It has two main input fields: 'Name' and 'Description'. The 'Name' field contains the text 'webserver'. The 'Description' field contains the text 'Webserver Security Group'. To the right of the 'Description' field, there is a block of text explaining that security groups are sets of IP filter rules applied to network settings for a VM, and that rules can be added after creation. At the bottom right, there are two buttons: 'Cancel' and 'Create Security Group'.

Create Security Group

Name *

webserver

Description *

Webserver Security Group

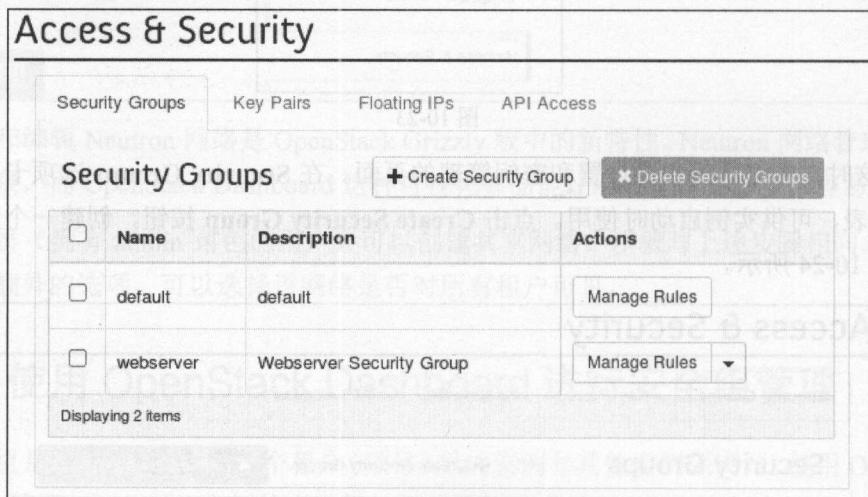
Description:

Security groups are sets of IP filter rules that are applied to the network settings for the VM. After the security group is created, you can add rules to the security group.

Cancel Create Security Group

图 10-25

3. 安全组创建成功后，所有可用的安全组都会显示在页面上。接下来，可以给新建的安全组添加网络安全规则，如图 10-26 所示。



The image shows the 'Access & Security' page. It has four tabs: 'Security Groups', 'Key Pairs', 'Floating IPs', and 'API Access'. The 'Security Groups' tab is active. Below the tabs, there are two buttons: '+ Create Security Group' and '* Delete Security Groups'. Below these buttons is a table with three columns: 'Name', 'Description', and 'Actions'. The table contains two rows: one for 'default' and one for 'webserver'. The 'webserver' row has a 'Manage Rules' button with a dropdown arrow. At the bottom left of the table, it says 'Displaying 2 items'.

Access & Security

Security Groups Key Pairs Floating IPs API Access

Security Groups + Create Security Group * Delete Security Groups

<input type="checkbox"/>	Name	Description	Actions
<input type="checkbox"/>	default	default	Manage Rules
<input type="checkbox"/>	webserver	Webserver Security Group	Manage Rules ▼

Displaying 2 items

图 10-26

编辑安全组添加和移除规则

要添加和移除规则或安全组，执行以下步骤。

1. 要创建一个新安全组或修改现有的安全组规则，点击该安全组的 **Manage Rules** 按钮，如图 10-27 所示。

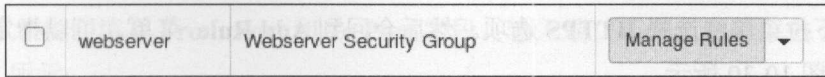


图 10-27

2. 点击 **Manage Rules** 按钮后，将看到现有规则的列表界面，如图 10-28 所示。

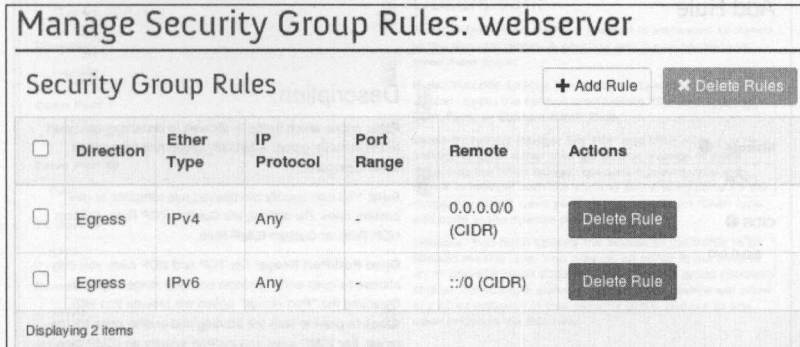


图 10-28

3. 要向这个新的安全组添加一条规则，点击 **Add Rule** 按钮。可以基于 ICMP、TCP、UDP 这 3 种协议来创建规则。此外还提供了一些常用服务的规则模板。本例将添加一个安全组规则以允许 HTTPS 从任何地方访问。要做到这一点，需要按图 10-29 所示的内容进行设置。

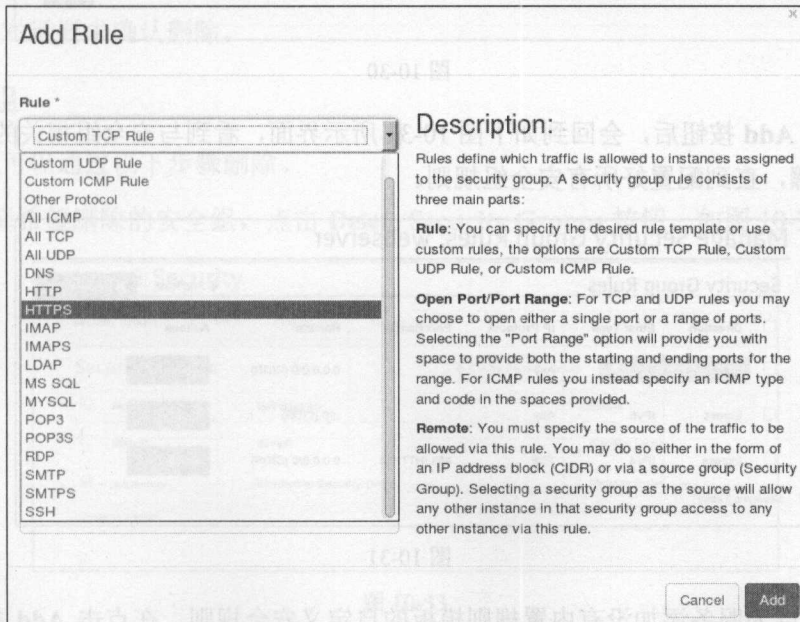


图 10-29

4. 从下拉菜单中选择 **HTTPS** 选项。然后会回到 **Add Rule** 菜单，可以指定网络流量的来源，如图 10-30 所示。

Add Rule

Add Rule

Rule *
HTTPS

Remote *
CIDR

CIDR *
0.0.0.0/0

Description:
Rules define which traffic is allowed to instances assigned to the security group. A security group rule consists of three main parts:
Rule: You can specify the desired rule template or use custom rules, the options are Custom TCP Rule, Custom UDP Rule, or Custom ICMP Rule.
Open Port/Port Range: For TCP and UDP rules you may choose to open either a single port or a range of ports. Selecting the "Port Range" option will provide you with space to provide both the starting and ending ports for the range. For ICMP rules you instead specify an ICMP type and code in the spaces provided.
Remote: You must specify the source of the traffic to be allowed via this rule. You may do so either in the form of an IP address block (CIDR) or via a source group (Security Group). Selecting a security group as the source will allow any other instance in that security group access to any other instance via this rule.

Cancel Add

图 10-30

5. 点击 **Add** 按钮后，会回到如下图 10-31 所示界面，看到与安全组相关的规则列表。重复上述步骤，直到配置好所有安全组规则。

Manage Security Group Rules: webserver

Security Group Rules + Add Rule X Delete Rules

<input type="checkbox"/>	Direction	Ether Type	IP Protocol	Port Range	Remote	Actions
<input type="checkbox"/>	Egress	IPv4	Any	-	0.0.0.0/0 (CIDR)	Delete Rule
<input type="checkbox"/>	Egress	IPv6	Any	-	:::0 (CIDR)	Delete Rule
<input type="checkbox"/>	Ingress	IPv4	TCP	443 (HTTPS)	0.0.0.0/0 (CIDR)	Delete Rule

Displaying 3 items

图 10-31

6. 也可以为服务添加没有内置规则模板的自定义安全规则。在点击 **Add** 按钮之后，从 **Rule** 下拉选项中选择 **Custom TCP Rule**。然后，从 **Open Port** 下拉选项中选择 **Port Range**

选项，会要求我们填写 **From Port** 和 **To Port** 字段。输入端口范围，然后点击 **Add** 按钮，如图 10-32 所示。

图 10-32

7. 注意，同样可以在第五步时移除规则。只需选择不再需要的规则，点击 **Delete Rule** 按钮，弹出对话要求确认删除。

删除安全组

安全组可以通过以下步骤删除。

1. 选择希望删除的安全组，点击 **Delete Security Groups** 按钮，如图 10-33 所示。

	Name	Description	Actions
<input type="checkbox"/>	default	default	Manage Rules
<input checked="" type="checkbox"/>	webservers	Webserver Security Group	Manage Rules

图 10-33

2. 会提示确认删除。点击 **OK** 按钮，移除安全组及相关访问规则，如图 10-34 所示。

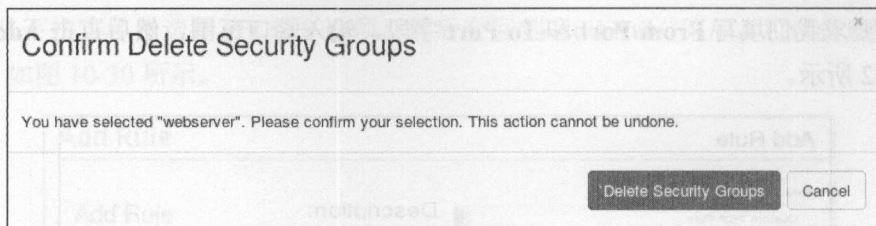


图 10-34



当一个与指定的安全组相关的实例正在运行时，将无法删除这个安全组。

工作原理

安全组对于 OpenStack 环境非常重要，它可以安全和一致地访问运行中的实例。用户通过创建、删除和修改安全组，可以为租户中创建一个安全的环境。

安全组只能在实例创建时才能进行关联，不能为一个已经运行实例添加一个新安全组，但是可以修改已分配给某个运行中实例的规则。例如，假设启动一个仅有默认安全组的实例。默认安全组设置只开放了 TCP 端口 22 和 ping 实例的权限。如果需要访问 TCP 端口 80，那么可以将此规则添加到默认的安全组或者重新启动实例分配一个新的安全组，以访问 TCP 端口 80。



安全组的修改会立即生效，任何分配该安全组的实例都会使用与它关联的新规则。

10.6 使用 OpenStack Dashboard 启动实例

使用 OpenStack Dashboard 启动实例非常简单。只需选择镜像、实例大小，然后启动。

准备工作

打开浏览器，指向 OpenStack Dashboard 地址 <http://192.168.100.200>，并以 demo 用户身份（1.6 节创建的）登录，密码为 openstack。

操作步骤

通过以下步骤，从 OpenStack Dashboard 启动一个实例。

1. 导航到 **Compute** 菜单下的 **Images** 选项卡，选择一个合适的镜像，如 `trusty-image` 服务器镜像，如图 10-35 所示。

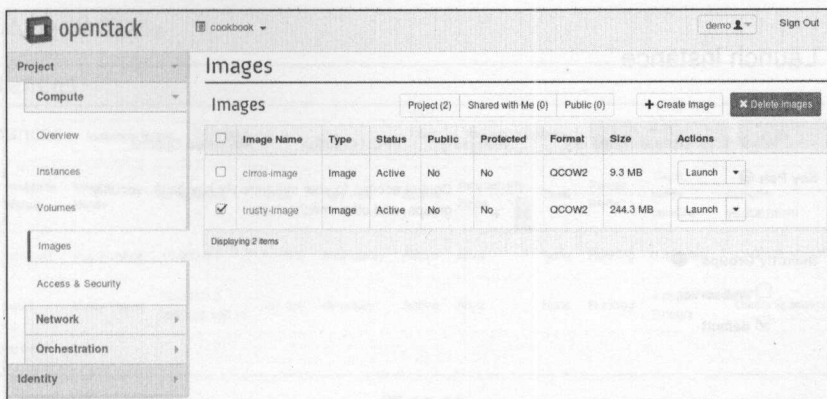


图 10-35

2. 点击启动镜像中的 **Actions** 下拉菜单中的 **Launch** 按钮。

3. 此时会出现一个提示对话框，需要输入实例名称（如 `horizon1`）。选择一个实例类型 `m1.tiny`，如图 10-36 所示。

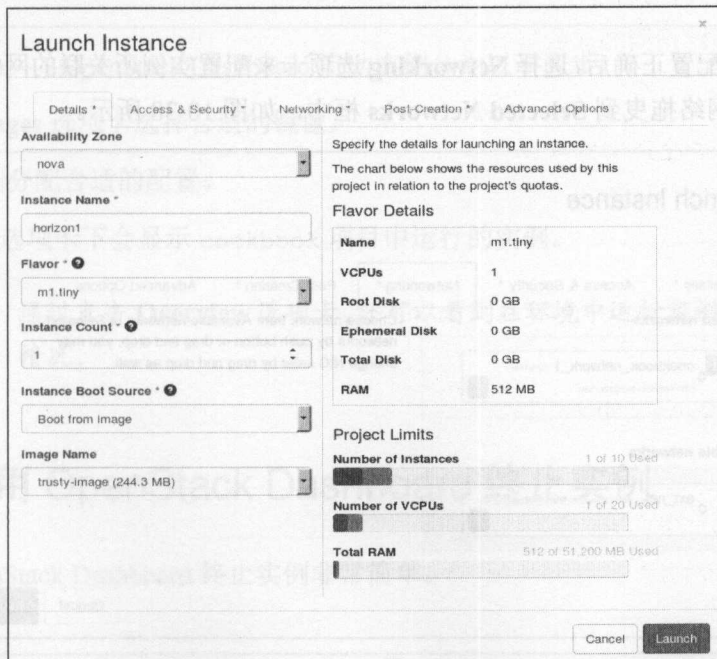


图 10-36

4. 接着打开 **Access & Security** 选项卡, 选择镜像所用的密钥对和安全组, 如图 10-37 所示。

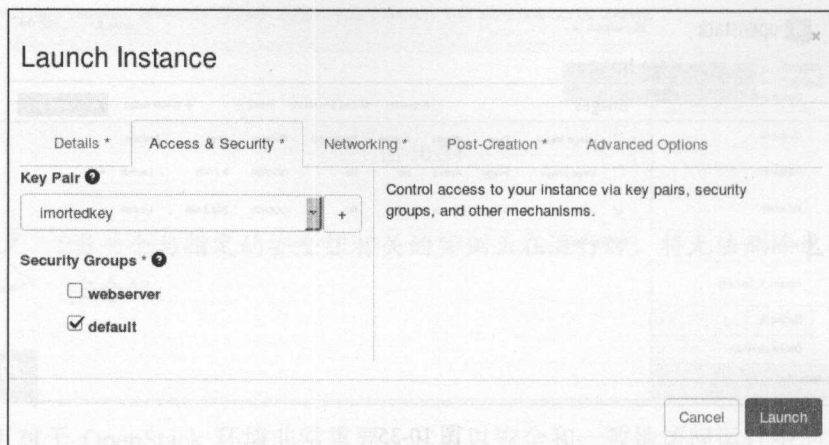


图 10-37



如果还没创建任何密钥对, 可以点击+按钮来导入一个密钥。

5. Neutron 配置正确后, 选择 **Networking** 选项卡来配置实例所关联的网络。将 **Available networks** 中的网络拖曳到 **Selected Networks** 框中, 如图 10-38 所示。

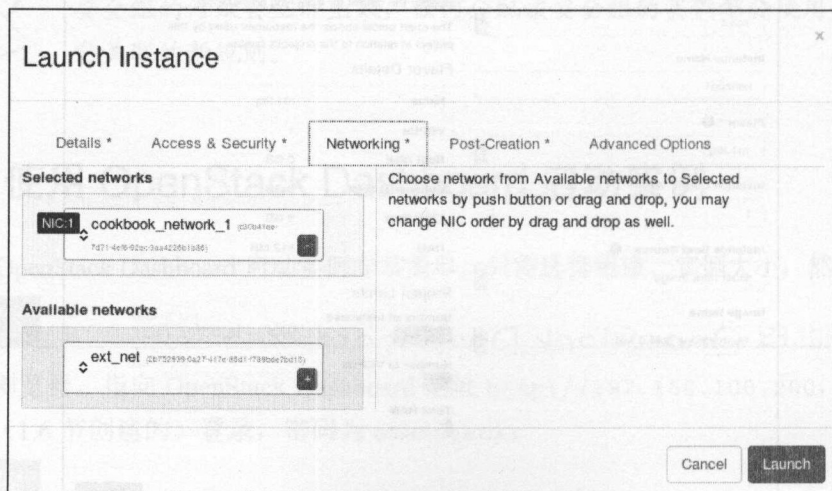


图 10-38

6. 选择完成后, 点击 **Launch** 按钮。

7. 这时, 返回到 **Instances** 选项卡, 此时实例会处于 **Build** 状态, 最后会变为 **Active** 状态, 如图 10-39 所示。

Instances											
Instances											
Instance Name		Instance Name	Filter	Filter	Launch Instance		Soft Reboot Instances		Terminate Instances		
<input type="checkbox"/>	Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
<input type="checkbox"/>	horizon1	trusty-image	10.200.0.4	m1.tiny	Importedkey	Active	nova	None	Running	0 minutes	Create Snapshot ▼
<input type="checkbox"/>	test1	trusty-image	10.200.0.2 192.168.100.11	m1.tiny	demokey	Active	nova	None	Running	4 days, 2 hours	Create Snapshot ▼
Displaying 2 items											

图 10-39

如果屏幕没有刷新, 可手动点击 **Instances** 选项卡刷新信息。

工作原理

从 Horizon——OpenStack Dashboard 启动实例分为以下两步。

1. 从 **Images** 选项卡选择合适的镜像。
2. 为实例分配合适的配置。

Instances 选项卡下会显示 cookbook 项目中运行的实例。

通过点击 **Overview** 选项卡, 还可以看到在环境中运行实例的大致情况。

10.7 使用 OpenStack Dashboard 终止实例

使用 OpenStack Dashboard 终止实例非常简单。

准备工作

打开浏览器, 指向 OpenStack Dashboard 地址 <http://192.168.100.200>, 并以 demo

用户身份（1.6 节创建的）登录，密码为 openstack。

操作步骤

要使用 OpenStack Dashboard 终止一个实例，需要执行以下步骤。

1. 打开 **Instances** 选项卡，使用实例名字边上的复选框选择想要终止的实例，然后点击红色的 **Terminate Instances** 按钮，如图 10-40 所示。

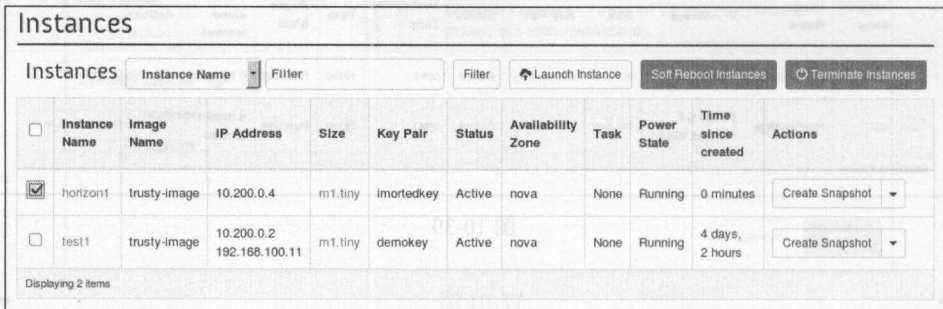


图 10-40

2. 在提示信息中，选择 **Terminate Instances** 终止所选择的实例，如图 10-41 所示。

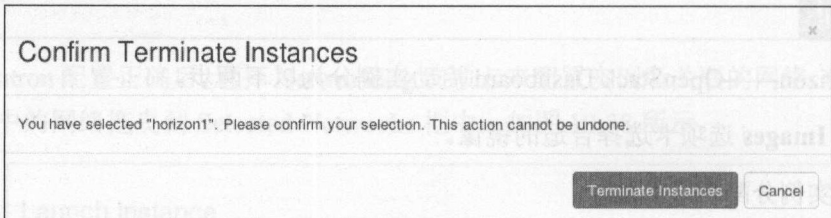


图 10-41

3. 在 **Instances** 页面中可以确认实例已成功终止。

工作原理

使用 OpenStack Dashboard 终止实例非常简单。只需选择运行的实例，点击 **Terminate Instances** 按钮，当某个实例已经选中时该按钮会高亮显示。点击 **Terminate Instances** 按钮之后，系统会提示确认终止该实例，以降低意外终止实例的风险。

10.8 使用 OpenStack Dashboard 连接到使用 VNC 的实例

OpenStack Dashboard 有一个非常方便的功能，它允许用户通过一个 VNC (Virtual

Network Console) 会话在从 Web 浏览器连接到正在运行的实例。这样, 无须额外单独调用一个 SSH 会话便可以管理虚拟机实例, 对于 Windows 操作系统来说是很好的方便访问远程桌面实例的特性。

准备工作

打开浏览器, 指向 OpenStack Dashboard 地址 `http://192.168.100.200`, 并以 demo 用户身份 (1.6 节创建的) 登录, 密码为 `openstack`。

操作步骤

- 要通过浏览器使用 VNC 链接到一个运行的实例, 需要执行以下步骤。
- 1. 选择 **Instances** 选项卡, 选择一个想要连接的实例。
 - 2. 在 **Create Snapshot** 按钮旁边有一个向下的箭头, 提示有更多的选择。点击后如图 10-42 所示。

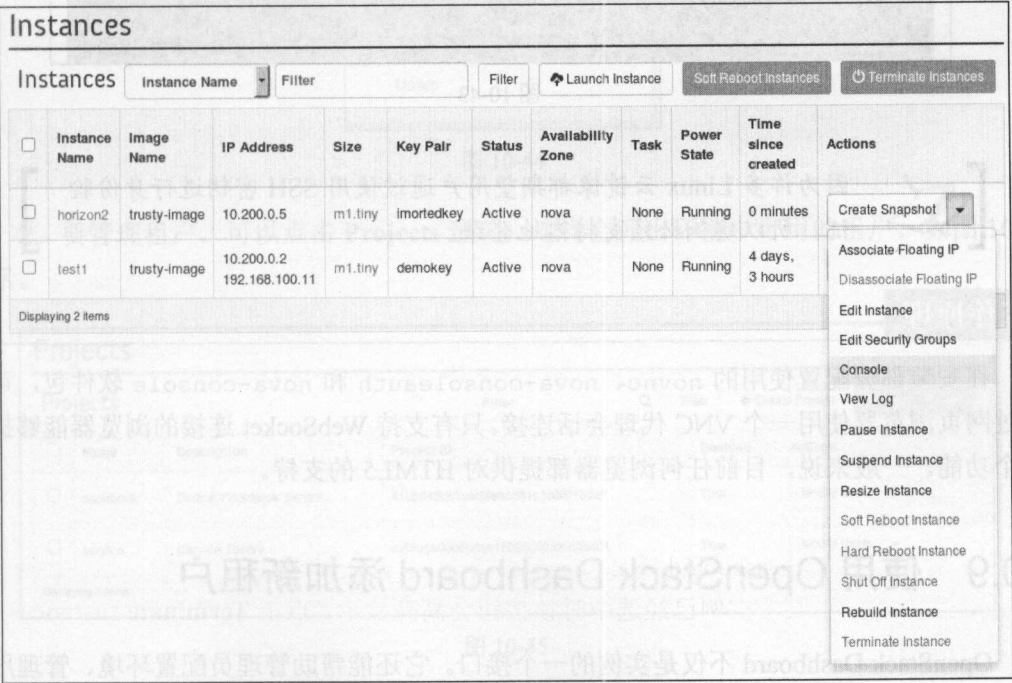


图 10-42

- 3. 选择 **Console** 选项。这将会显示出一个控制台屏幕, 可以让用户登录到实例中, 如图 10-43 所示。

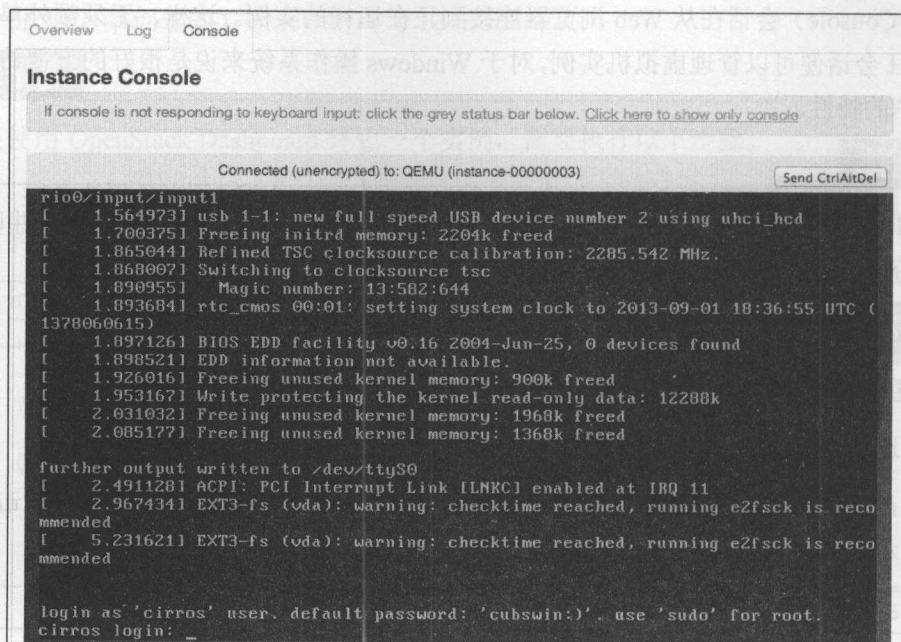


图 10-43



因为许多 Linux 云镜像都期望用户通过使用 SSH 密钥进行身份验证，所以实例必须支持本地登录。

工作原理

在安装部分配置使用的 novnc、nova-consoleauth 和 nova-console 软件包，可以通过网页浏览器使用一个 VNC 代理会话连接。只有支持 WebSocket 连接的浏览器能够提供这个功能。一般来说，目前任何浏览器都提供对 HTML5 的支持。

10.9 使用 OpenStack Dashboard 添加新租户

OpenStack Dashboard 不仅是实例的一个接口。它还能帮助管理员配置环境、管理用户和租户。

在 OpenStack Dashboard 中，租户也被称为项目。租户可以添加成员，而在 OpenStack Dashboard 中添加新租户的操作非常简单。

准备工作

打开浏览器，指向 OpenStack Dashboard 地址 <http://192.168.100.200>，并以 demo 用户身份（1.6 节创建的）登录，密码为 openstack。

操作步骤

要给 OpenStack 环境添加一个租户，需要执行以下步骤。

1. 当以管理员用户身份登录进来时，会在 **Identity** 选项卡下看到更多的菜单选项，其中就包含 **Projects** 选项，如图 10-44 所示。

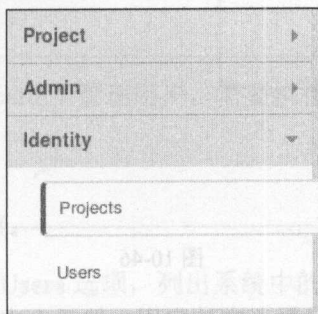


图 10-44

2. 要管理租户，可以点击 **Projects** 选项，这样会列出环境中可用的租户，如图 10-45 所示。

Projects					
Filter		Q	Filter	+ Create Project	* Delete Projects
<input type="checkbox"/>	Name	Description	Project ID	Enabled	Actions
<input type="checkbox"/>	cookbook	Default Cookbook Tenant	412b512d9fca40feacc81c1b831f83a9	True	Modify Users ▼
<input type="checkbox"/>	service	Service Tenant	a2f9ec6666d4ee1bb25859fbcc3b401	True	Modify Users ▼
Displaying 2 items					

图 10-45

3. 点击 **Create Project** 按钮，创建一个新的租户。

4. 然后提交一个表单，填写租户名称和描述信息。输入 horizon 作为租户名，并输入一个描述，如图 10-46 所示。

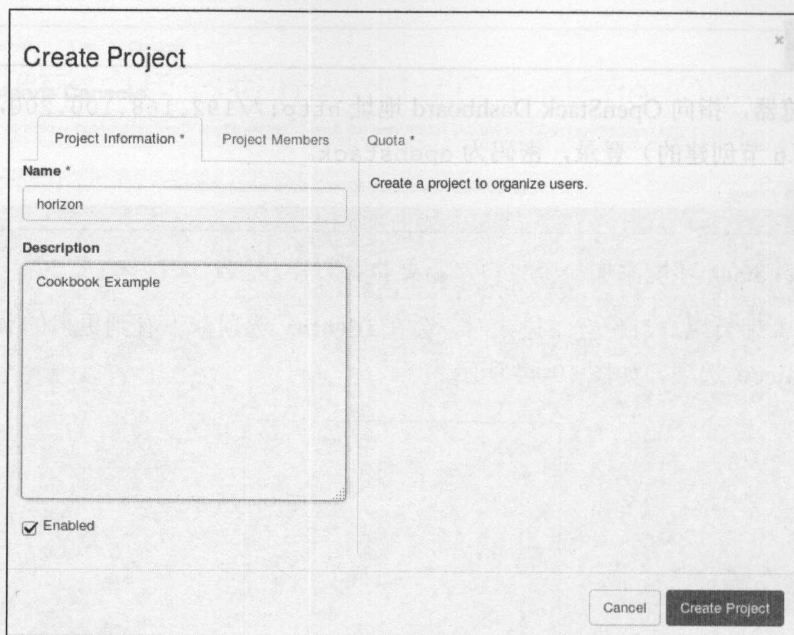
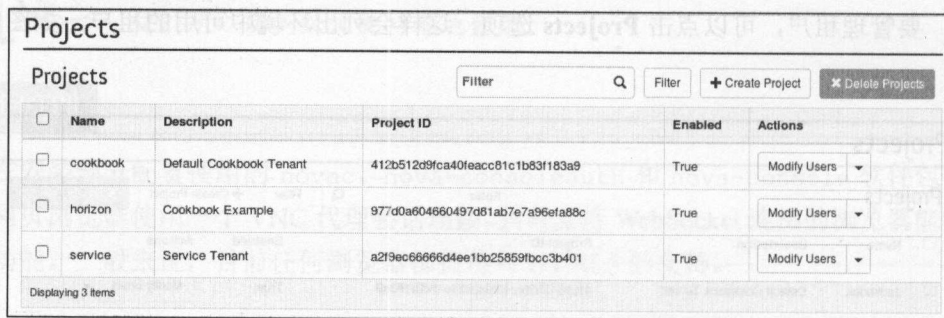


图 10-46

5. 选中 **Enabled** 复选框，确保租户被启用，然后点击 **Create Project** 按钮。这时将列出可用的租户名单，提示 horizon 租户已成功创建，如图 10-47 所示。



Projects					
<div>Filter <input type="text"/></div> <div>Filter <input type="text"/></div> <div>+ Create Project</div> <div>✖ Delete Projects</div>					
<input type="checkbox"/>	Name	Description	Project ID	Enabled	Actions
<input type="checkbox"/>	cookbook	Default Cookbook Tenant	412b512d9fca40feacc81c1b83f183a9	True	Modify Users
<input type="checkbox"/>	horizon	Cookbook Example	977d0a604860497d81ab7e7a96efa88c	True	Modify Users
<input type="checkbox"/>	service	Service Tenant	a2f9ec6666d4ee1bb2559fbcc3b401	True	Modify Users

Displaying 3 items

图 10-47

工作原理

OpenStack Dashboard 是一个功能丰富的界面，为管理 OpenStack 环境提供了命令行工具之外的有益补充。可以简单地在 OpenStack Dashboard 里创建一个用户所属的租户（Ubuntu 的界面使用的是项目）。在创建租户时，需要以管理员用户身份登录，才能看到完整的租户管理界面。

10.10 使用 OpenStack Dashboard 进行用户管理

OpenStack Dashboard 可以通过 Web 界面进行用户管理。这使得管理员能轻松地创建和修改 OpenStack 环境内的用户。要管理用户，必须以 admin 角色用户登录。

准备工作

打开浏览器，指向 OpenStack Dashboard 地址 `http://192.168.100.200`，并以 demo 用户身份（1.6 节创建的）登录，密码为 `openstack`。

操作步骤

要完成在 OpenStack Dashboard 下管理用户，需要执行以下步骤。

添加用户

要添加用户，执行下列步骤。

- 1. 在 Identity 面板中点击 **Users** 选项，列出系统中的用户列表，如图 10-48 所示。

Users

Users

Filter

Q

Filter

+ Create User

✕ Delete Users

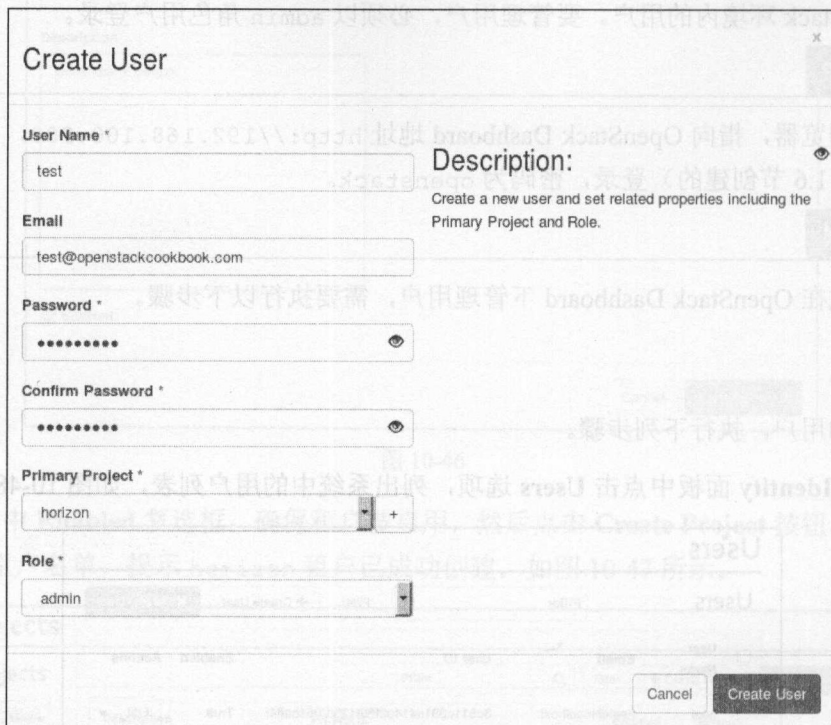
<input type="checkbox"/>	User Name	Email	User ID	Enabled	Actions
<input type="checkbox"/>	heat	heat@localhost	3c511c391e414d2f86f125f2061ca84f	True	Edit <div></div>
<input type="checkbox"/>	demo	demo@localhost	447df476321449869d5a0b2c12cb4943	True	Edit <div></div>
<input type="checkbox"/>	nova	nova@localhost	4da2cb3596374437bbd9cf4c32948c47	True	Edit <div></div>
<input type="checkbox"/>	cellometer	heat@localhost	58013fd4cc7847bdba81d0271b7c7f38	True	Edit <div></div>
<input type="checkbox"/>	cinder	cinder@localhost	661847bc877c4caab004f326297dd92b	True	Edit <div></div>
<input type="checkbox"/>	keystone	keystone@localhost	95c813c530fb40b9a65afcc36561b8a0	True	Edit <div></div>
<input type="checkbox"/>	admin	root@localhost	9e89da2972524ca7bd0a331471c0128c	True	Edit <div></div>
<input type="checkbox"/>	glance	glance@localhost	cdbb6a51ac984dde977e4d5175ec1d11	True	Edit <div></div>
<input type="checkbox"/>	neutron	neutron@localhost	d31814ad50c740699580136cf19b91b8	True	Edit <div></div>

Displaying 9 items

图 10-48

2. 要创建一个新用户，选择 **Create User** 按钮。

3. 会出现填写一个关于用户名的表格。输入用户名称、用户电子邮件地址和用户密码。本例创建了一个名为 `test` 的用户，设置 `openstack` 作为密码，并分配该用户给 `horizon` 租户，如图 10-49 所示。



Create User

User Name *
test

Email
test@openstackcookbook.com

Password *
••••••••

Confirm Password *
••••••••

Primary Project *
horizon

Role *
admin

Description:
Create a new user and set related properties including the Primary Project and Role.

Cancel Create User

图 10-49

4. 返回到 OpenStack 环境的用户列表界面，此时会收到一个用户创建成功的信息。

删除用户

要删除用户，执行下列步骤。

1. 在 **Identity** 面板中点击 **Users** 选项列出系统中的用户列表。

2. 此时会看到 OpenStack 环境中设置的用户列表。点击想要删除的用户的 **Edit** 按钮，出现一个下拉菜单，选择 **Delete User** 选项，如图 10-50 所示。

3. 选择之后会弹出来一个确认对话框。点击 **Delete User** 按钮，会从系统中删除该用户，如图 10-51 所示。

<input type="checkbox"/>	nova	nova@localhost	4da2cb3596374437bbd8cf4c32648c47	True	Edit
<input type="checkbox"/>	test	test@openstackcookbook.com	55228e55989f4986a821a254c55f013d	True	Edit
<input type="checkbox"/>	ceilometer	heat@localhost	58013fd4cc7647bdba81d0271b7c7135	True	Disable User Delete User
<input type="checkbox"/>	cinder	cinder@localhost	661847bc877c4caab004f326297dd92b	True	Edit

图 10-50

Confirm Delete User

You have selected "test". Please confirm your selection. This action cannot be undone.

Delete User
Cancel

图 10-51

更新用户信息和密码

要更新用户信息和密码，执行下列步骤。

1. 在 **Identity** 面板中点击 **Users** 选项，列出系统中的用户列表。
2. 要修改用户密码、用户电子邮件地址或所属项目（租户），可点击该用户的 **Edit** 按钮。
3. 选择之后会弹出来一个相关信息对话框。填完相关信息之后，点击 **Update User** 按钮，如图 10-52 所示。

Update User

User Name *

Email

Password

Confirm Password

Primary Project *

horizon

Description:

Edit the user's details, including the Primary Project and Role.

Cancel

Update User

图 10-52

为租户添加用户

要为租户添加用户，执行下列步骤。

1. 在 **Identity** 面板点击 **Projects** 选项，列出系统中的租户列表，如图 10-53 所示。

Projects

Projects

Filter

Q

Filter

+ Create Project

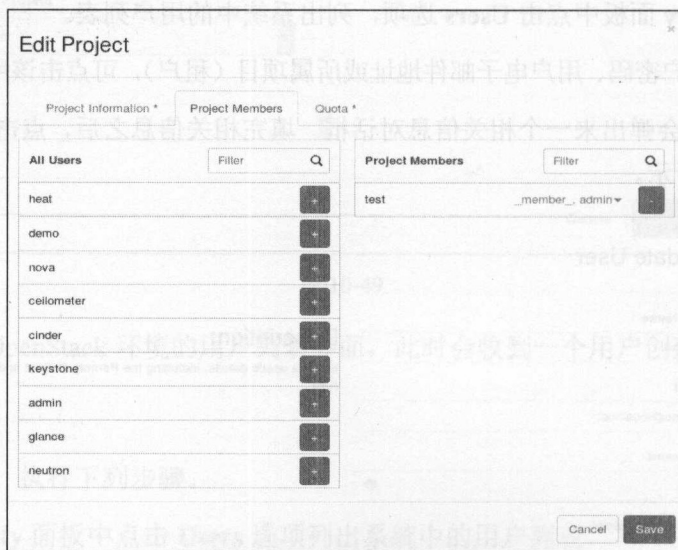
X Delete Projects

<input type="checkbox"/>	Name	Description	Project ID	Enabled	Actions
<input type="checkbox"/>	cookbook	Default Cookbook Tenant	412b512d9fca40feacc81c1b83f183a9	True	<div>Modify Users</div>
<input type="checkbox"/>	horizon	Cookbook Example	977d0a604660497d81ab7e7a96efa85c	True	<div>Modify Users</div>
<input type="checkbox"/>	service	Service Tenant	a2f9ec6666d4ee1bb25859bcc3b401	True	<div>Modify Users</div>

Displaying 3 items

图 10-53

2. 点击 **Modify Users** 选项，弹出一个与租户关联的用户列表，以及可以添加到该租户的用户列表，如图 10-54 所示。



Edit Project

Project Information *
Project Members
Quota *

All Users

Filter Q

heat	+
demo	+
nova	+
cellometer	+
cinder	+
keystone	+
admin	+
glance	+
neutron	+

Project Members

Filter Q

test	_member_ ▾	admin
------	------------	-------

Cancel
Save

图 10-54

3. 要为该列表添加一个新的用户，只需点击+（加号）按钮。
4. 要改变用户在租户中的角色，可点击用户名旁边的下拉列表并选择一个新角色，如图 10-55 所示。

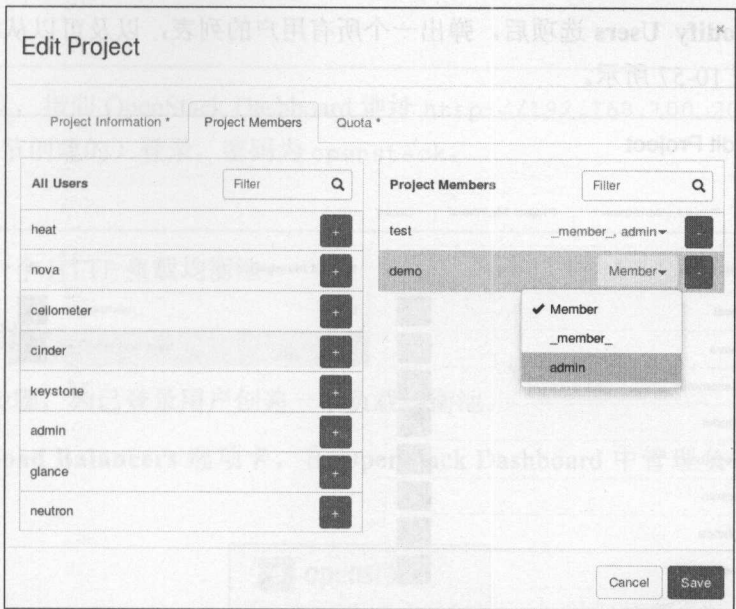


图 10-55

5. 点击对话框底部的 **Save** 按钮，将看到租户信息已更新的消息。登录时，用户可以在不同的租户中启动实例了。

从租户中删除用户

要从租户中删除用户，需要执行下列步骤。

1. 在 **Identity** 面板中点击 **Projects** 选项列出系统中的租户列表。
2. 要删除租户中的一个用户，如 horizon，点击 **Modify Users** 按钮，如图 10-56 所示。

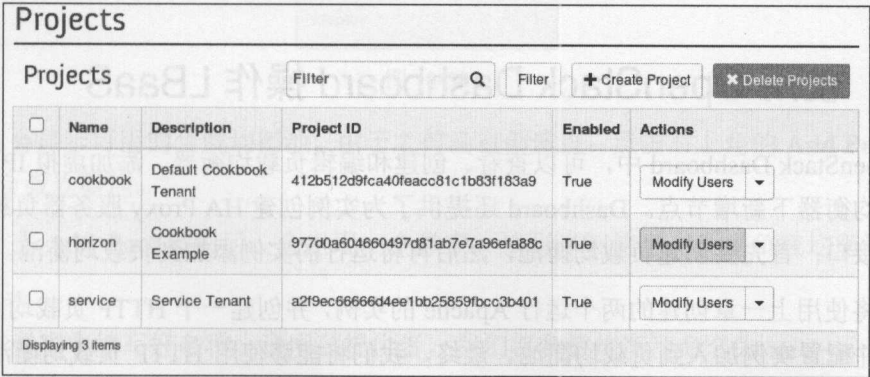


图 10-56

3. 点击 **Modify Users** 选项后，弹出一个所有用户的列表，以及可以从租户中删除的用户列表，如图 10-57 所示。

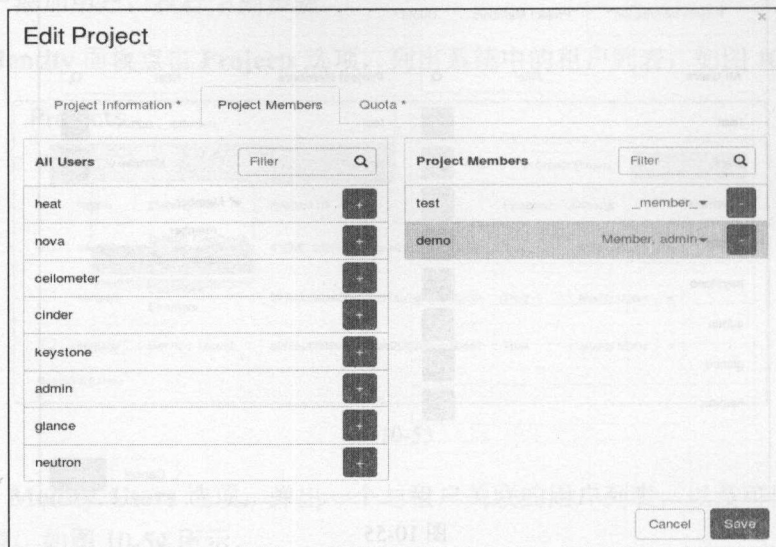


图 10-57

4. 要从租户中删除一个用户，点击该用户旁边的-（减号）按钮。
5. 点击对话框底部的 **Save** 按钮，将看到租户信息已更新的消息。

工作原理

OpenStack Dashboard 有一个功能丰富的界面，为管理 OpenStack 云环境提供了命令行工具之外的有益补充。该界面为管理员提供了尽可能直观的功能。不但可以很容易地创建用户、修改租户内成员、更新密码，还可以从系统中完全删除它们。

10.11 使用 OpenStack Dashboard 操作 LBaaS

在 OpenStack Dashboard 中，可以查看、创建和编辑负载均衡器，添加虚拟 IP (VIP)，并在负载均衡器下新增节点。Dashboard 还提供了为实例创建 HA Proxy 服务器负载均衡服务的用户接口；首先要创建负载均衡池，然后再将运行的实例添加到负载均衡池。

本节将使用上一章创建的两个运行 Apache 的实例，并创建一个 HTTP 负载均衡池、一个 VIP，并配置实例加入到负载均衡池。最终，我们将能够使用 HTTP 负载均衡池地址，向运行 Apache 的两个实例传输数据。

准备工作

打开浏览器，指向 OpenStack Dashboard 地址 `http://192.168.100.200`，并以 demo 用户身份（1.6 节创建的）登录，密码为 `openstack`。

操作步骤

首先创建一个 HTTP 负载均衡池。

创建负载均衡池

执行如下步骤，为已登录用户创建一个负载均衡池。

1. 选择 **Load Balancers** 选项卡，在 OpenStack Dashboard 中管理负载均衡器，如图 10-58 所示。

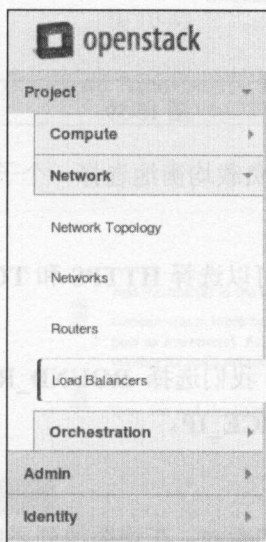


图 10-58

2. 这会显示可用的负载均衡池。由于之前没有创建过，点击右上角的 **Add Pool** 按钮，添加一个负载均衡池。

3. 点击 **Add Pool** 按钮后，会出现一个弹窗。填写详细信息，新建负载均衡池，如图 10-59 所示。

4. 在弹窗中填写好名称、说明和提供者。我们将负载均衡池命名为 `web-pool`，并填写合适的说明。由于要创建一个 HA Proxy，选择默认的提供者。

Add Pool

Add New Pool *

Name *
web-pool

Description
Web Pool

Provider
haproxy (default)

Subnet *
10.200.0.0/24

Protocol *
HTTP

Load Balancing Method *
ROUND_ROBIN

Admin State *
UP

Create Pool for current project.
Assign a name and description for the pool. Choose one subnet where all members of this pool must be on. Select the protocol and load balancing method for this pool. Admin State is UP (checked) by default.

Cancel Add

图 10-59

5. 点击 **Subnet** 下拉菜单，为负载均衡池选择一个子网。所有的实例都连接到了私有网络，所以选择 10.200.0.0/24。

6. 选择 **HTTP** 协议，当然也可以选择 **HTTPS** 和 **TCP** 协议，具体要根据运行的应用类型决定。

7. 选择希望使用的路由算法；我们选择 **ROUND_ROBIN** 均衡方法。其他选项包括 **LEAST_CONNECTIONS** 和 **SOURCE_IP**。

8. 维持 **Admin State** 为 **UP**。

9. 点击 **Add** 按钮，创建负载均衡池。在均衡池列表中应该会出现新建的负载均衡池，如图 10-60 所示。

Load Balancer

PoolsMembersMonitors

Pools

+ Add Pool✖ Delete Pools

<input type="checkbox"/>	Name	Description	Provider	Subnet	Protocol	Status	VIP	Actions
<input type="checkbox"/>	web-pool	Web Pool	haproxy	10.200.0.0/24	HTTP	ACTIVE	-	<div>Edit Pool ▾</div>

Displaying 1 item

图 10-60

添加池成员

按照如下步骤，向负载均衡中添加实例。

1. 添加负载均衡池后，应该还停留在 **Load Balance** 页面。点击 **Members** 选项卡，会看到活跃成员列表（如果有）或一个空列表，如图 10-61 所示。

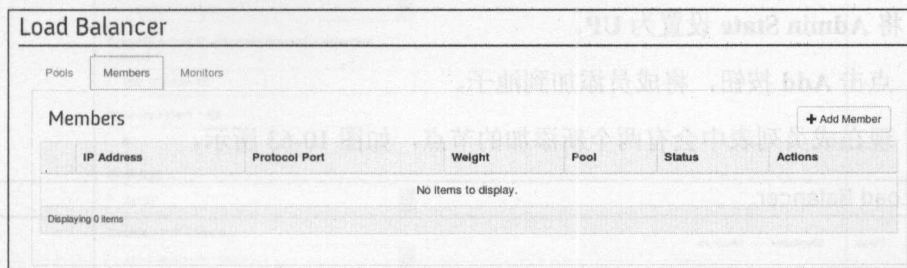


图 10-61

2. 在 **Members** 选项卡中，点击 **Add Member** 按钮。会出现如图 10-62 所示菜单。

图 10-62

3. 选择刚才创建的负载均衡池 **web-pool**，然后指定要加入该池子的成员或实例。如果没有运行 **Apache** 的实例，请参照 9.2 节所示操作创建。
4. 选择负载均衡池中成员的权重。在本例中，池子中的两个成员权重相同，所以设置权重为 **1**。
5. 选中的协议端口将用于访问所有成员，由于使用的是 **HTTP** 协议，我们将端口设置为 **80**。将 **Admin State** 设置为 **UP**。
6. 点击 **Add** 按钮，将成员添加到池子。
7. 现在成员列表中会有两个新添加的节点，如图 10-63 所示。

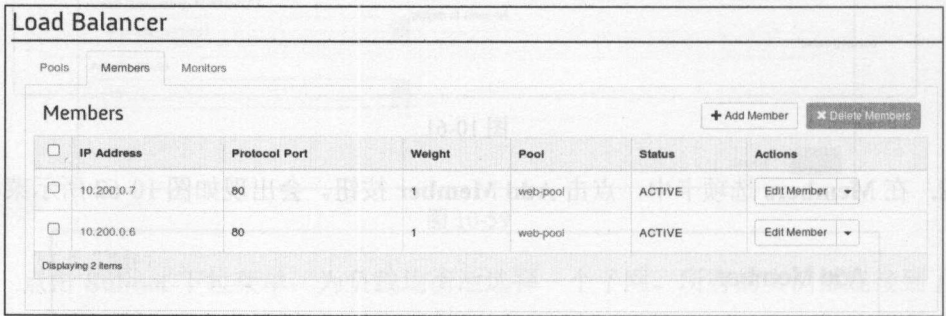


图 10-63

添加 VIP 至负载均衡池

在外部网络上创建 **VIP**，将支持访问负载均衡池及背后的实例。执行如下步骤，创建 **VIP**。

1. 从 **Load Balancer** 页面，选择 **Pools** 选项卡，点击 **Edit Pool** 按钮旁边的下拉箭头。这时会出现一个下拉菜单，包含添加 **VIP** 的选项，如图 10-64 所示。

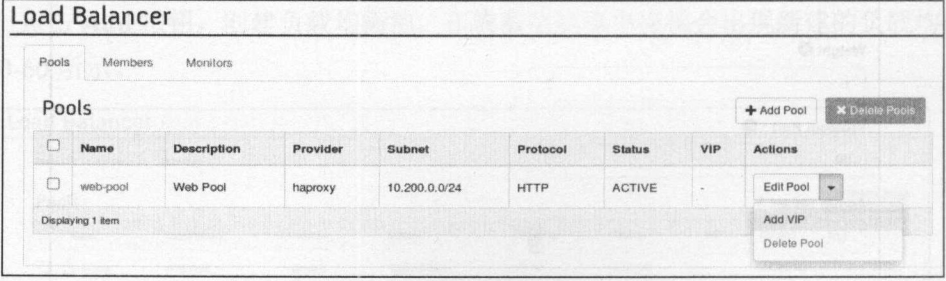


图 10-64

2. 点击 **Add VIP** 选项，会出现一个新建 **VIP** 的弹窗，如图 10-65 所示。

Add VIP

Specify VIP *

Name *

web-vip

Description

Web VIP

VIP Subnet

192.168.100.0/24

Specify a free IP address from the selected subnet *

192.168.100.12

Protocol Port *

80

Protocol *

HTTP

Session Persistence

No Session Persistence

Connection Limit

-1

Admin State *

UP

Create a VIP for this pool. Assign a name, description, IP address, port, and maximum connections allowed for the VIP. Choose the protocol and session persistence method for the VIP. Admin State is UP (checked) by default.

Cancel

Add

图 10-65

- 3. 输入 VIP 的名称和描述。
- 4. 对于 **VIP Subnet**，选择外部子网 192.168.100.0/24，接下来是子网中的一个可用 IP。我们选择 192.168.100.12。
- 5. 将 **Protocol Port** 设置为 80，然后选择 **Protocol** 为 HTTP。
- 6. 如果不希望为 VIP 设置最大连接数，将 **Connection Limit** 设置为-1。
- 7. 点击 **Add** 按钮，创建 VIP。这会创建一个新的 VIP，会出现在当前负责均衡池列表下，如图 10-66 所示。

Load Balancer

Pools

Members

Monitors

+

 Add Pool

✖

 Delete Pools

<input type="checkbox"/>	Name	Description	Provider	Subnet	Protocol	Status	VIP	Actions
<input type="checkbox"/>	web-pool	Web Pool	haproxy	10.200.0.0/24	HTTP	ACTIVE	web-vip	<div>Edit Pool</div>

Displaying 1 item

图 10-66

8. 现在再点击 **web-pool** 时, 会看到如图 10-67 所示的信息。

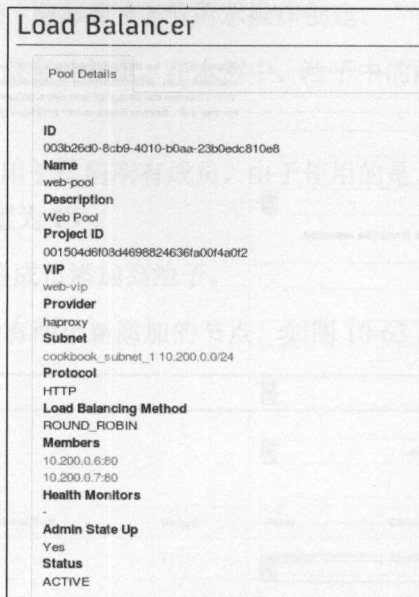


图 10-67

9. 点击详细信息页面中的 **web-vip** 链接, 查看 VIP 的具体情况, 如图 10-68 所示。

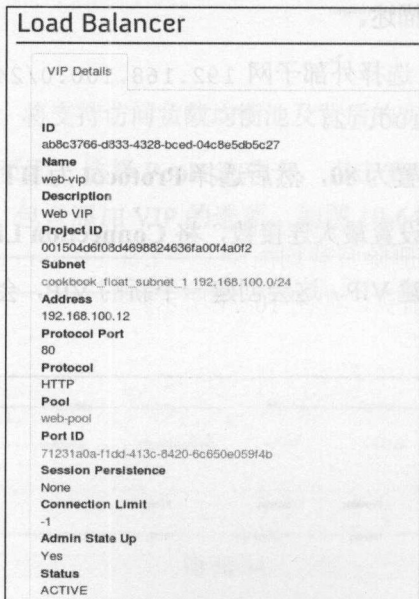


图 10-68

10. 你可以在浏览器中输入 VIP 的地址，测试负载均衡是否生效。如果选择 **ROUND_ROBIN** 作为路由算法，每次刷新浏览器都会访问一个不同的节点。

删除负载均衡器

要删除负载均衡器，首先需要删除添加的 VIP，然后删除负载均衡池。

1. 从 **Load Balancer** 页面，选中 **Pools** 选项卡，然后点击 **Edit Pool** 按钮旁边的下拉箭头。这会出现一个下拉菜单，包含删除 VIP 的选项，如图 10-69 所示。

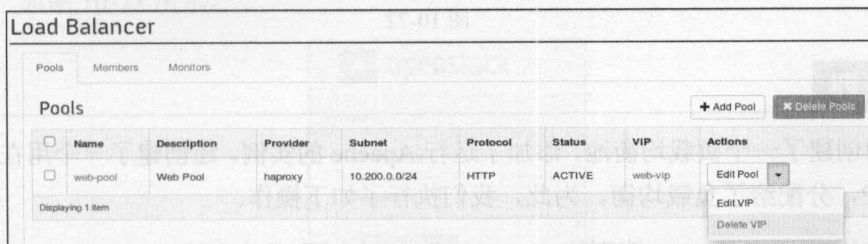


图 10-69

2. 选择 **Delete VIP** 下拉选项，会出现一个警告，请求确认是否要删除 VIP。点击 **Delete VIP** 按钮确认，如图 10-70 所示。

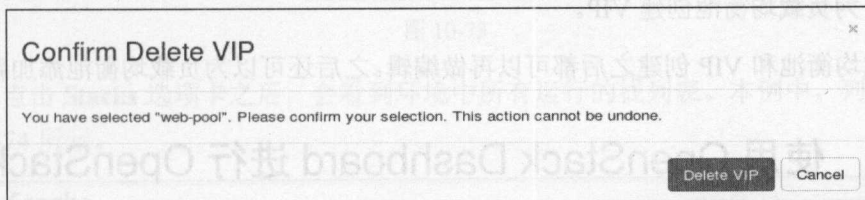


图 10-70

3. 删除 VIP 后，可以删除负载均衡池。从 **Load Balancer** 页面的 **Pools** 选项卡，点击 **Edit Pool** 按钮旁边的下拉箭头，如图 10-71 所示。

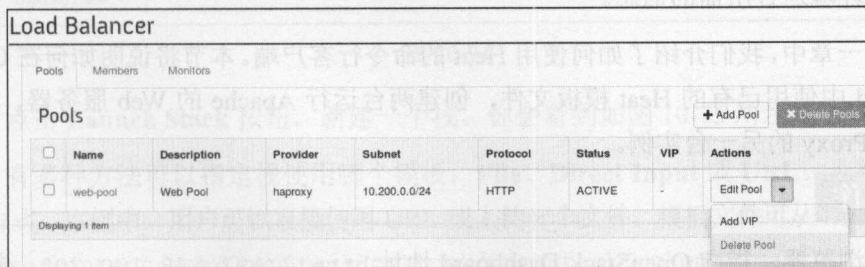


图 10-71

4. 选择下拉列表中的 **Delete Pool** 选项，删除负载均衡池。这时会要求确认是否删除。如果准备好删除负载均衡池，点击该按钮，如图 10-72 所示。

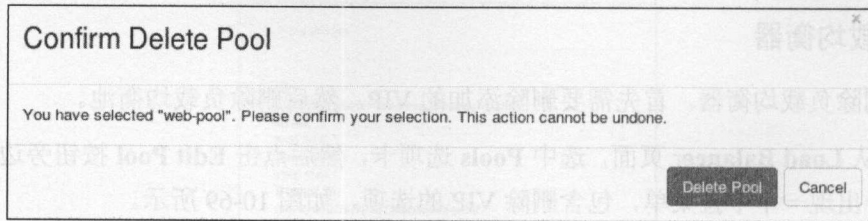


图 10-72

工作原理

我们创建了一个负载均衡池，添加了运行 Apache 的实例。还创建了一个用在外部网络的虚拟 IP，分配给了负载均衡。为此，我们执行了如下操作。

- (1) 从 **Load Balancer** 页面的 **Pools** 选项卡下创建负载均衡池。
- (2) 创建负载均衡池时选择所有节点连接的子网。
- (3) 添加成员到负载均衡池。
- (4) 为负载均衡池创建 VIP。

负载均衡池和 VIP 创建之后都可以再做编辑。之后还可以为负载均衡池添加新的成员。

10.12 使用 OpenStack Dashboard 进行 OpenStack 编排

Heat 是 OpenStack 的编排引擎，可以让用户通过模板快速启动完整的环境。Heat 模板，也称为 Heat Orchestration Templates (HOT)，是基于 YAML (Yet Another Markup Language) 编写的文件。模板文件描述了环境中使用的资源，实例的类型和大小，实例连接的网络，以及其他环境运行所需的信息。

在上一章中，我们介绍了如何使用 Heat 的命令行客户端。本节将说明如何在 OpenStack Dashboard 中使用已有的 Heat 模板文件，创建两台运行 Apache 的 Web 服务器，并连接到运行 HA Proxy 的另一台实例。

准备工作

打开浏览器，指向 OpenStack Dashboard 地址 <http://192.168.100.200>，并以 demo 用户身份（1.6 节创建的）登录，密码为 openstack。

操作步骤

首先，我们要在 OpenStack Dashboard 中启动栈（stack）。

启动栈

要为已登录的用户启动 Heat 栈，需执行以下步骤。

1. 选择 **Orchestration** 菜单下的 **Stacks** 选项卡，查看 OpenStack Dashboard 中可用的 Heat 栈，如图 10-73 所示。

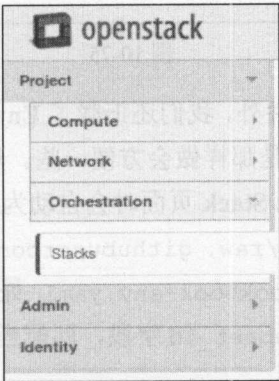


图 10-73

2. 点击 **Stacks** 选项卡之后，会看到环境中所有运行的栈列表。本例中，列表是空的，如图 10-74 所示。

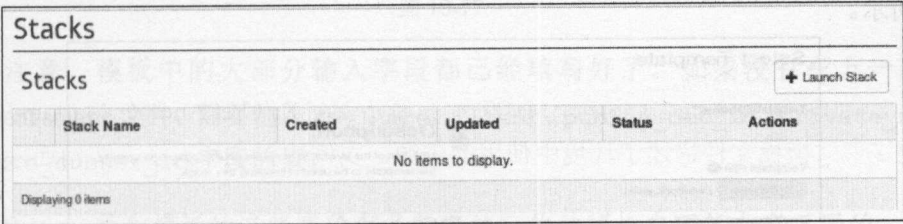


图 10-74

3. 点击 **Launch Stack** 按钮，新建一个栈。你会看到如图 10-75 所示的窗口。

4. 有多种方法可以指定栈使用哪个模板：**File**、**Direct Input** 或 **URL**。选择对自己最方便的方式。本例中，用户可以直接使用 **URL** 或上传一个文件。模板文件可从 <https://raw.githubusercontent.com/OpenStackCookbook/OpenStackCookbook/master/cookbook.yaml> 处下载。我们从系统中上传文件。

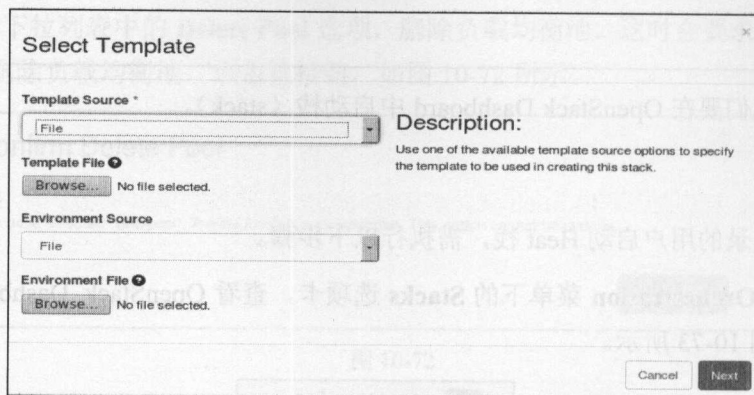


图 10-75

5. 除了 `cookbook.yaml` 文件外，我们还下载了 **Environment Source** 文件。在本例中，我们不一定要使用这个文件，但是那样做会方便一些。环境文件中存储了要手动在浏览器中输入的值，在第 8 步的 **Launch Stack** 页面时会自动为我们加载这些值。在本例中，我们使用的环境文件可以从 <https://raw.githubusercontent.com/OpenStackCookbook/OpenStackCookbook/master/cookbook-env.yaml> 处下载。更新 `public_net_id`, `private_net_id` 和 `private_subnet_id` 字段，匹配当前环境的设置。



如果不确定到哪查找网络信息，请参考 10.3 节。

6. 选择好 **Template Source** 和 **Environment Source** 文件之后，点击 **Next** 按钮，如图 10-76 所示。

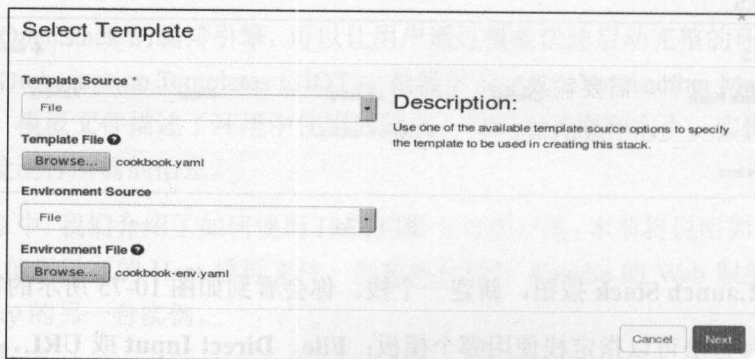


图 10-76

7. 我们的实例环境文件中包含如下代码。

parameters:

```
key_name: demokey
image: trusty-image
flavor: m1.tiny
public_net_id: 5e5d24bd-9d1f-4ed1-84b5-0b7e2a9a233b
private_net_id: 25153759-994f-4835-9b13-bf0ec77fb336
private_subnet_id: 4cf2c09c-b3d5-40ed-9127-ec40e5e38343
```

8. 点击 **Next**, 会出现 **Launch Stack** 窗口, 包含了所有的输入字段, 如图 10-77 所示。

Launch Stack

Stack Name *
haproxy101

Description:
Create a new stack with the provided values.

Creation Timeout (minutes) *
60

☐ Rollback On Failure

Password for user "admin" *

key_name *
demokey

image *
trusty-image

public_net_id *
5e5d24bd-9d1f-4ed1-84b5-0b7e2a9a233b

private_net_id *
25153759-994f-4835-9b13-bf0ec77fb336

flavor *
m1.tiny

private_subnet_id *
4cf2c09c-b3d5-40ed-9127-ec40e5e38343

Cancel **Launch**

图 10-77

9. 注意, 模板中的大部分输入字段都已经填写好了。如果没有在上一步指定 `environment source` 文件, 需要输入 `key_name`、`flavor`、`public_net_id`、`private_net_id` 和 `private_subnet_id` 字段的值。



每个模板的这些字段都是特定的。你的模板中可能会有不同的字段。

10. 输入栈名称以及用户的密码。如果以 `admin` 或 `demo` 用户身份登录, 密码就是 `openstack`。

11. 点击 **Launch** 按钮开始创建栈。如果所有输入字段都正确, 会看到栈正在创建的提示, 如图 10-78 所示。

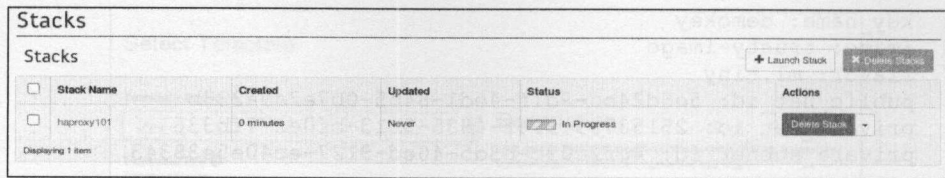


图 10-78

12. 栈创建完成后，如果过程中没有出错，会发现栈的状态更新为了 **Complete**，如图 10-79 所示。

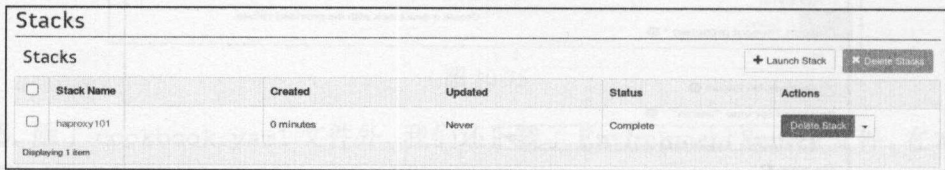


图 10-79

查看栈详情

启动栈之后，会出现很多相关信息，包括输入、输出，如果出错了还会有为什么栈创建失败的说明信息。

1. 在 **Stacks** 列表中点击栈的名称，查看栈的详情。第一个视图是网络拓扑，如图 10-80 所示。

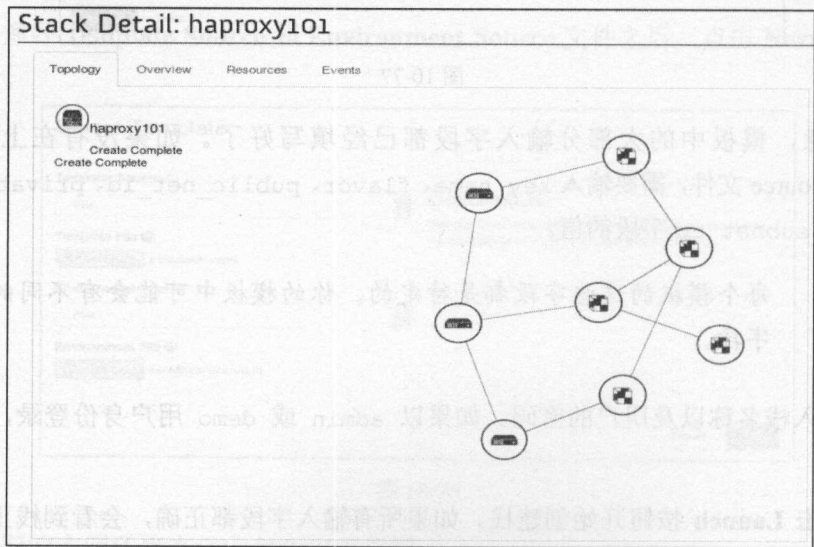


图 10-80

点击相应节点，探索拓扑的具体细节。如果图表无法完全展开或希望从不同的角度看，可以在窗口上拖动图表。

2. **Stack Detail** 中的下一个选项卡，将提供创建栈时用到所有信息，如图 10-81 所示。

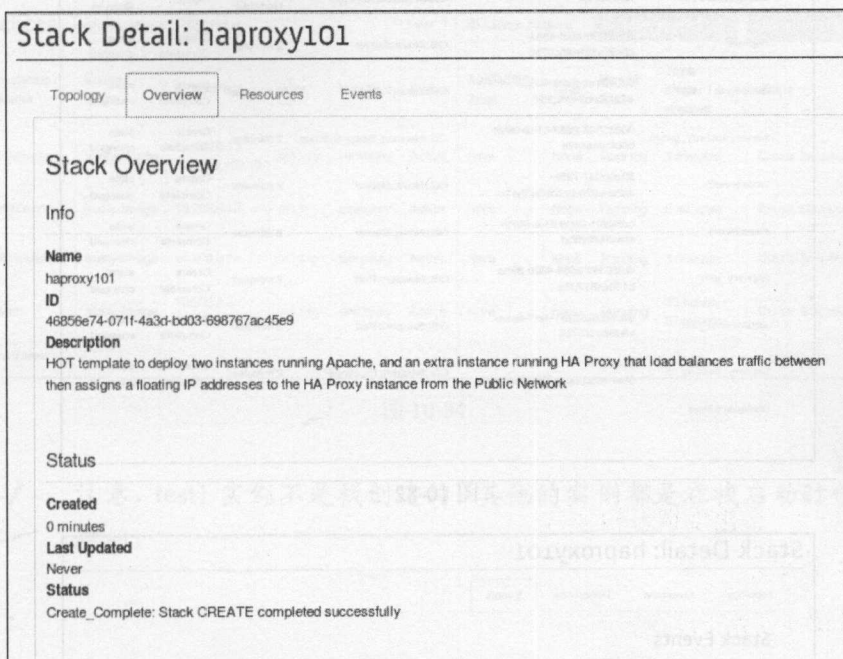


图 10-81

Overview 选项卡中包含的栈信息如下：

- 信息；
- 状态；
- 输出；
- 栈参数；
- 启动参数。

3. **Resource** 选项卡显示的是栈启动过程中创建的所有 HEAT 资源，如图 10-82 所示。

如果栈启动时出错，请检查该页面，查看哪个组件创建失败。

4. **Events** 选项卡会显示栈创建时发生的所有事件。这个页面在 Heat 模板排错时也非常有用，如图 10-83 所示。

Stack Detail: haproxy101					
<div>Topology Overview Resources Events</div>					
Stack Resources					
Stack Resource	Resource	Stack Resource Type	Date Updated	Status	Status Reason
haproxy	55fd3e3b-b03f-4894-b5c6-a161ca5cb7ac	OS::Nova::Server	2 minutes	Create Complete	state changed
webserver1_port	027e51aa-6e29-406a-a731-4eb3434c2df5	OS::Neutron::Port	2 minutes	Create Complete	state changed
server_security_group	908a0368-2233-41d2-9892-b8bfcdae4494	OS::Neutron::SecurityGroup	2 minutes	Create Complete	state changed
webserver2	25e6b3d7-728a-403a-b076-0abab563be1c	OS::Nova::Server	2 minutes	Create Complete	state changed
webserver1	cd4a191-3add-41df-b972-afac3475605d	OS::Nova::Server	2 minutes	Create Complete	state changed
haproxy_port	4bf85340-b688-4320-96ba-b190a4877d6e	OS::Neutron::Port	2 minutes	Create Complete	state changed
webserver2_port	0ec3dd9d-28a1-4ef7-8eae-c9d69c4757b6	OS::Neutron::Port	2 minutes	Create Complete	state changed
haproxy_floating_ip	084496f-59ef-4383-8c34-9f629e41d1f9	OS::Neutron::FloatingIP	2 minutes	Create Complete	state changed
Displaying 8 items					

图 10-82

Stack Detail: haproxy101				
<div>Topology Overview Resources Events</div>				
Stack Events				
Stack Resource	Resource	Time Since Event	Status	Status Reason
haproxy	-	2 minutes	Create Complete	state changed
haproxy	-	2 minutes	Create In Progress	state changed
webserver1	-	2 minutes	Create Complete	state changed
webserver2	-	2 minutes	Create Complete	state changed
haproxy_floating_ip	-	2 minutes	Create Complete	state changed
webserver1	-	2 minutes	Create In Progress	state changed
webserver2	-	2 minutes	Create In Progress	state changed
haproxy_floating_ip	-	2 minutes	Create In Progress	state changed
haproxy_port	-	2 minutes	Create Complete	state changed
webserver1_port	-	2 minutes	Create Complete	state changed
webserver2_port	-	2 minutes	Create Complete	state changed
haproxy_port	-	2 minutes	Create In Progress	state changed
webserver1_port	-	2 minutes	Create In Progress	state changed
webserver2_port	-	2 minutes	Create In Progress	state changed
server_security_group	-	2 minutes	Create Complete	state changed
server_security_group	-	2 minutes	Create In Progress	state changed
Displaying 16 items				

图 10-83

5. 在 Heat 栈运行时，还可以在 **Compute** 选项卡的 **Instances** 选项中看到栈创建了多少个实例。图 10-84 是 **Instances** 页面的大致情况。

Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
<input type="checkbox"/> HAProxy	trusty-image	10.200.0.13 192.168.100.15	m1.tiny	demokey	Active	nova	None	Running	4 minutes	Create Snapshot
<input type="checkbox"/> Webserver1	trusty-image	10.200.0.12	m1.tiny	demokey	Active	nova	None	Running	5 minutes	Create Snapshot
<input type="checkbox"/> Webserver2	trusty-image	10.200.0.11	m1.tiny	demokey	Active	nova	None	Running	5 minutes	Create Snapshot
<input type="checkbox"/> test1	trusty-image	10.200.0.2 192.168.100.11	m1.tiny	demokey	Active	nova	None	Running	22 hours, 37 minutes	Create Snapshot

Displaying 4 items

图 10-84



注意，test1 实例不是栈创建的。其他的实例都是在栈启动时创建的。

删除栈

删除栈很简单；不过，也会删除栈启动时创建的所有资源。根据如下步骤操作。

1. 要删除栈，首先在 **Stacks** 页面查看可用的栈，如图 10-85 所示。

Stack Name	Created	Updated	Status	Actions
<input type="checkbox"/> haproxy101	1 hour, 25 minutes	Never	Complete	Delete Stack

Displaying 1 item

图 10-85

2. 点击相应的 **Delete Stack** 按钮删除栈，会要求确认是否删除，如图 10-86 所示。

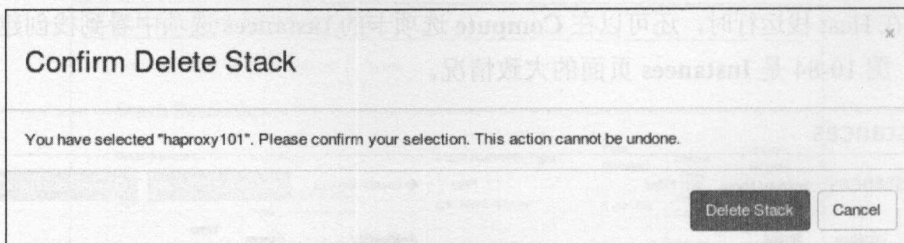


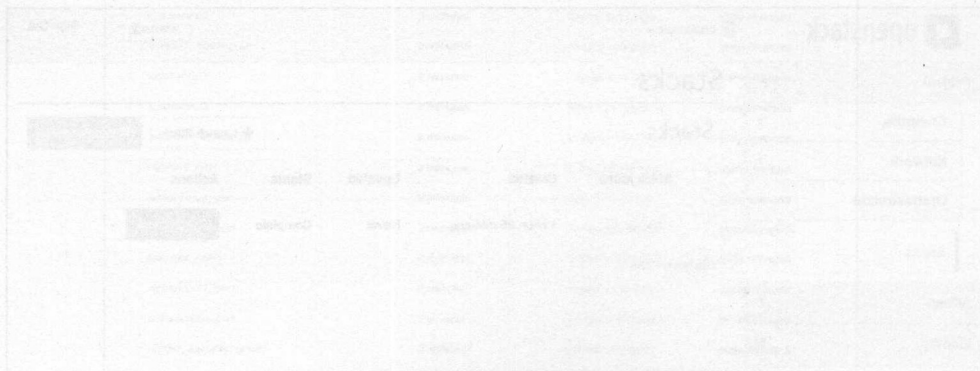
图 10-86

3. 确认删除之后，所有栈相关的资源将被销毁。

工作原理

我们使用 OpenStack Dashboard 创建、查看和删除了编排栈。首先需要从 GitHub 下载 HA Proxy 的 Heat 编排模板文件。因为我们用了一个环境文件，还要修改相应的输入。你自己的模板可能包含不同的字段。

在启动 HA Proxy 栈之后，我们介绍了它的网络拓扑、使用的资源和创建时发生的事件。栈启动时创建的资源还会体现在环境的其他部分。如果要启动新的实例，可以在 **Instances** 页面找到所有可用的实例。建议只从 OpenStack Dashboard 的 **Orchestration** 部分或命令行，删除和修改栈启动时创建的资源。从 Dashboard 删除栈会销毁所有相关的资源。



第 11 章

生产环境中的 OpenStack

本章将讲述以下内容：

- 安装 MariaDB Galera 集群
- MariaDB Galera 集群配置 HA Proxy
- 配置 HA Proxy 实现高可用
- 使用 Corosync 安装并配置 Pacemaker
- 使用 Pacemaker 和 Corosync 配置 OpenStack 服务
- 绑定多个网卡实现高冗余
- 使用 Ansible 自动安装 OpenStack — 主机配置
- 使用 Ansible 自动安装 OpenStack — Playbook 配置
- 使用 Ansible 自动安装 OpenStack — Playbook 运行

11.1 简介

OpenStack 是一整套的软件系统，提供了可伸缩的云环境，能够部署在遍布全球的数据中心里。管理远程地点中的软件安装与本地安装软件不同（有时更具挑战），因此开发了许多工具和技巧来协助完成该任务。在运行环境中，必须要考虑如何处理软件和硬件故障等问题。发现单点故障（Single Points of Failure, SPOF）并提供更多增加其弹性的方法，将确保出线问题时我们的 OpenStack 环境保持可用。

本章将介绍一些方法和软件，协助管理数据中心等生产环境中的 OpenStack 环境，从让服务变得高可用，到自动化安装以实现一致性和可重复性。

11.2 安装 MariaDB Galera 集群

OpenStack 支持多个数据库后端，最常见的是 MySQL 或其开源分支 MariaDB。可以通过多种方式，提高 MariaDB 的弹性和可用性。以下方法使用一个负载均衡来作为 Galera 多读/写的前端，负载同步备份。Galera 是针对 MariaDB InnoDB 数据库的同步多主集群。Galera 支持在所有节点上进行同步数据写入操作，任何一个节点均可以完全活动/活动的形式负责写入操作。它还具备自动节点管理功能，即自动从集群移除错误节点，并注册新节点。其优势是，如果出现数据库节点故障，系统将具备更强的弹性，因为每个节点都存储了数据副本。Galera 由奇数个节点组成，在出现节点故障时这个设计会起作用。Galera 会在剩余节点中进行法定人数投票，来决定状态。投票要求多数通过，即只有两个节点的情况下不能自动恢复。这是因为一个节点故障之后，会使剩余的节点自动进行非主状态。偶数节点组成的集群可能会出现势均力敌的情况，如果集群的某个地方出现网络连接问题，导致分为两个数量相同的分区，而且每个分区都无法达到数量要求，会进入非主状态。

准备工作

确保有 3 台运行 Ubuntu 14.04 的服务器，至少一个用于访问机器并配置 Galera 备份的接口。参照 <https://github.com/OpenStackCookbook/MariaDB-Galera> 上的指南，启动一个合适的 Vagrant 环境。下面的操作步骤中，也会详细介绍。

操作步骤

本节中，我们将在 3 个节点上安装 MariaDB 和 Galera，3 个节点分别叫 Galera1、Galera2 和 Galera3。每个节点均会在 172.16.0.0/16 网络上分配一个单独的 IP 地址，各自机器上的网络接口均设为 eth1。以下步骤中用到的 IP 地址为 172.16.0.191、172.16.0.192 和 172.16.0.193。3 个节点均运行 Ubuntu 14.04 LTS 系统。具体步骤如下：

1. 在第一台服务器 Galera1 上，配置 apt 通过如下命令获取 MariaDB 集群 10.0 软件包。

```
apt-get install software-properties-common
sudo apt-key adv --recv-keys --keyserver \
    hkp://keyserver.ubuntu.com:80 0xc9cb082a1bb943db
sudo add-apt-repository 'deb
http://lon1.mirrors.digitalocean.com/mariadb/rep0/10.0/ubuntu
trusty main'
sudo apt-get update
```


2. 运行如下命令，安装软件包。

```
DEBIAN_FRONTEND=noninteractive apt-get install \
    rsync galera mariadb-galera-server
```

3. 执行如下命令，确保 MariaDB 没有运行。

```
sudo service mysql stop
```

4. 在 Galera2 和 Galera3 服务器上重复 1~3 步。

5. 然后配置 MariaDB 使用 Galera。编辑/etc/mysql/conf.d/galera.conf 文件，加入如下内容。

```
[mysqld]
# mysql settings
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
query_cache_size=0
query_cache_type=0
bind-address=0.0.0.0
# galera settings
wsrep_provider=/usr/lib/galera/libgalera_smm.so
wsrep_cluster_name="my_wsrep_cluster"
wsrep_cluster_address="gcomm://172.16.0.191,172.16.0.192,172.16.0.193"
wsrep_sst_method=rsync
```

6. 登录 Galera1 执行如下命令，启动一个新的 Galera 备份集群。

```
sudo service mysql start --wsrep-new-cluster
```

7. 登录 Galera2 和 Galera3，在每个节点上执行如下命令。

```
sudo service mysql start
```

8. 测试 MySQL 服务是否启动。登录任意节点，运行如下命令：

```
mysql -u root -e 'SELECT VARIABLE_VALUE as "cluster size" \
    FROM INFORMATION_SCHEMA.GLOBAL_STATUS \
    WHERE VARIABLE_NAME="wsrep_cluster_size"'
```

上述命令的输出如图 11-1 所示。

+	+
	cluster size
+	+
	3
+	+

图 11-1

工作原理

我们对 3 台运行 Ubuntu 14.04 的服务器进行了配置，安装了 MariaDB 和 Galera。这步

是通过将 MariaDB 的代码库添加到 apt 环境，并安装适当的程序包来实现的。

在 3 个节点上完成上述步骤后，我们编辑 `/etc/mysql/conf.d/galera.conf` 文件，配置 MariaDB 使用 Galera。这个文件中有一部分描述如何使用 Galera，我们指定集群的名称、集群中使用的 IP 地址，以及数据备份的方式。

```
wsrep_provider=/usr/lib/galera/libgalera_smm.so
wsrep_cluster_name="my_wsrep_cluster"
wsrep_cluster_address="gcomm://172.16.0.191,172.16.0.192,172.16.0.193"
wsrep_sst_method=rsync
```

之后，启动集群。为此，登录一个节点，执行如下命令。

```
service mysql start --wsrep-new-cluster
```

然后，使用如下命令，正常启动其余节点。

```
service mysql start
```

11.3 MariaDB Galera 集群配置 HA Proxy

配置好 MariaDB Galera 集群后，每个节点均可处理数据流量，数据写入操作会无缝地备份到集群的其他节点。我们可以在配置文件中任意使用 MariaDB 节点地址，如果该节点出错，将无法连接到数据库，OpenStack 环境也无法使用。一个比较好的解决方案是使用负载均衡作为 MariaDB 集群的前端。考虑到任何一个节点都可完成读取操作，保证数据一致性，负载均衡是一个很好的方案。一般来说，推荐如 F5 或 Brocade 等提供的物理负载均衡。没有物理负载均衡的情况下，可以使用高可用代理（High Availability Proxy）。

准备工作

安装两台服务器，均运行 Ubuntu 14.04 操作系统，配置到与 OpenStack 环境和 MariaDB 集群相同的管理网络下。在以下步骤中，这两个节点将位于 172.16.0.248 和 172.16.0.249 两个 IP 地址。在下一节中，将配置这两个节点使用浮动 IP 地址（通过 keepalived 设置）172.16.0.251。在 OpenStack 配置文件中配置数据库连接时将用到该浮动 IP。

操作步骤

由于要设置相同的服务器组成一对，我们先配置一台服务器，然后对第二台重复相同的操作。第一台将使用 172.16.0.248 这个 IP 地址。然后使用 172.16.0.249 重复配置过程。

为了给 MariaDB Galera 负载均衡配置 HA Proxy，重复第 1 步至第 5 步两次，创建两个 HA Proxy 实例，均配置为可访问 MariaDB Galera 节点。

1. 首先使用 apt-get 安装 HA Proxy, 命令如下。

```
sudo apt-get update
sudo apt-get install haproxy
```

2. 安装好 HA Proxy 之后, 对第一台代理服务器进行简单配置, 方便访问 MariaDB Galera 集群。为此, 编辑/etc/haproxy/haproxy.cfg 文件, 然后插入如下内容。

```
global
    log 127.0.0.1 local0
    log 127.0.0.1 local1 notice
    maxconn 4096
    user haproxy
    group haproxy
    daemon

defaults
    log global
    mode http
    option tcplog
    option dontlognull
    retries 3
    option redispatch
    maxconn 4096
    timeout connect 50000ms
    timeout client 50000ms
    timeout server 50000ms

# MySQL Load Balance Pool
listen mysql 0.0.0.0:3306
    mode tcp
    balance roundrobin
    option tcpka
    option mysql-check user haproxy
    server mysql1 172.16.0.191:3306 weight 1
    server mysql2 172.16.0.192:3306 weight 1
    server mysql3 172.16.0.193:3306 weight 1
```

3. 保存并退出文件。然后使用如下命令启动 HA Proxy。

```
sudo sed -i 's/^ENABLED.*/ENABLED=1/' /etc/defaults/haproxy
sudo service haproxy start
```

4. 在使用 HA Proxy 服务器访问 3 个 MariaDB 节点之前, 必须创建 haproxy.cfg 文件中指定的用户, 这个文件用来检查 MariaDB 是否正在运行。为此, 在集群中添加一个可以连接到 MariaDB 的用户。在任意 Galera 节点上, 使用 mysql 客户端创建用户 haproxy, 不设置密码。该用户可以从 HA Proxy 服务器的 IP 地址访问数据库, 运行如下命令。

```
mysql -u root -h localhost \
    -e "GRANT ALL ON *.* to haproxy@'172.16.0.248';"
```

5. 接着执行一个类似的操作, 在 Galera 集群中添加一个 root 用户, 可以运行来自 HA Proxy 服务器的 MySQL 命令。


```
mysql -u root -h localhost \
-e "GRANT ALL ON *.* to root@'172.16.0.248' IDENTIFIED BY
'openstack' WITH GRANT OPTION;"
```



重复第 1 步到第 5 步，将 172.16.0.248 这个 IP 地址替换为第二个节点的地址 172.16.0.249。

工作原理

HA Proxy 是一个非常流行、有用的代理以及负载均衡，是作为 MariaDB 集群前端，为其提供负载均衡能力的理想选择。设置起来也比较简单。

第一个要求是监听对应的端口，MariaDB 使用的是 3306。配置文件中有关监听设置的那行，将 0.0.0.0 设置为了监听地址，意味着将监听所有地址，不过可以指定一个具体的地址，来加强对环境中访问的控制。

为了使用 MariaDB，必须设置为 tcp 模式。我们将 keepalived 设置为 tcpka，确保某个客户端连接到 MariaDB 服务器时长连接不会被打断。

负载均衡的具体方式是轮询算法（Round Robin），非常适合多主集群、任意节点可读可写的场景。

我们添加了一个基础检查，确保 MariaDB 服务器会合理地标记是否下线。由于使用了内建的 mysql-check 选项（要求 MariaDB 中设置一个可登录、退出 MariaDB 节点的用户），当某台 MariaDB 服务器故障时，会绕过该服务器，转发流量到主节点的一台 MariaDB 服务器。注意，这一步不会检查某个具体的表是否存在，当然这可以通过在每个节点上执行检查脚本来实现，但是配置更加复杂。

配置 HA Proxy 的最后一步，是列出可以监听的节点和地址，它们组成了负载均衡池。

11.4 配置 HA Proxy 实现高可用

上节中配置了一个双节点的 HA Proxy，可用作 OpenStack 配置文件中的 MariaDB 端点。只使用单个 HA Proxy 作为高可用多主集群的负载均衡，不是推荐做法，因为负载均衡会造成单点故障。为了解决这个问题，可以按照并配置 keepalived，这样就能在多个 HA Proxy 服务器之间共享一个浮动 IP 地址。我们可以使用这个浮动 IP 地址，来保障 OpenStack 服务。

准备工作

以 root 用户身份，登录上节中创建的两个 HA Proxy 服务器。

操作步骤

由于已经设置了两个相同的 HA Proxy 服务器（一个位于 172.16.0.248，另一个位于 172.16.0.249），我们将制定一个浮动虚拟 IP 地址 172.16.0.251，可以绑定到一台服务器，并在其故障时切换到另一台。为此，执行下操作。

1. 只有一台 HA Proxy 服务器作为多主 MariaDB 集群的前端，会使 HA Proxy 服务器成为单点故障。为了解决这个问题，我们使用 keepalived 提供的虚拟路由冗余协议（Virtual Router Redundancy Protocol）管理功能。需在两台 HA Proxy 服务器上安装 keepalived。与之前一样，我们先配置一台服务器，然后在第二台上重复操作。具体命令如下。

```
sudo apt-get update
sudo apt-get install keepalived
```

2. 为了将运行的软件绑定至服务器上并不实际存在的地址，在 sysctl.conf 文件添加一个选项。在 /etc/sysctl.conf 文件中添加如下行。

```
net.ipv4.ip_nonlocal_bind=1
```

3. 执行如下命令，使得改动生效。

```
sudo sysctl -p
```

4. 现在可以配置 keepalived。为此，创建 /etc/keepalived/keepalived.conf 文件，写入如下内容。

```
vrp_script chk_haproxy {
    script "killall -0 haproxy" # verify the pid exists or
    not
    interval 2                # check every 2 seconds
    weight 2                   # add 2 points if OK
}

vrp_instance VI_1 {
    interface eth1             # interface to monitor
    state MASTER
    virtual_router_id 51       # Assign one ID for this router
    priority 101                # 101 on master, 100 on backup
    virtual_ipaddress {
        172.16.0.251          # the virtual IP
    }
    track_script {
        chk_haproxy
    }
}
```

5. 现在执行如下命令，在这台服务器上启动 keepalived。

```
sudo service keepalived start
```

6. 第一个 HA Proxy 服务器上现在已经开始运行 keepalived，这台服务器也被指定为主节点，然后我们在第二个服务器上重复上面的步骤，只需要对 keepalived.conf 文件做两处改动即可（state 设置为 BACKUP，priority 设置为 100），完整内容如下。

```

vrrp_script chk_haproxy {
    script "killall -0 haproxy" # verify the pid exists or
not
    interval 2                # check every 2 seconds
    weight 2                   # add 2 points if OK
}

vrrp_instance VI_1 {
    interface eth1             # interface to monitor
    state BACKUP
    virtual_router_id 51       # Assign one ID for this router
    priority 100                # 101 on master, 100 on backup
    virtual_ipaddress {
        172.16.0.251           # the virtual IP
    }
    track_script {
        chk_haproxy
    }
}

```

7. 在第二个节点上启动 keepalived，两个节点就会进入联动状态。因此，如果第一个 HA Proxy 服务器关机，第二个服务器就会切换为浮动 IP 地址 172.16.0.251。两秒后，就可以无干扰地域 MariaDB 集群建立新的连接。可通过如下命令，连接到数据库集群，测试 HA Proxy 和 MariaDB Galera 集群设置是否正常工作。

```
mysql -uroot -popenstack -h 172.16.0.251
```

8. 为了检查 keepalived 是否正常工作，在每个节点上查看 /var/log/syslog 下的信息。执行如下命令。

```
sudo grep VRRP /var/log/syslog
```

在当前绑定浮动 IP 地址的节点上，会看到如图 11-2 所示输出。

```

Jul 2 16:06:46 packer-vmware-iso Keepalived[3869]: Starting VRRP child process, pid=3873
Jul 2 16:06:46 packer-vmware-iso Keepalived_vrrp[3873]: VRRP_Script(chk_haproxy) succeeded
Jul 2 16:06:47 packer-vmware-iso Keepalived_vrrp[3873]: VRRP_Instance(VI_1) Transition to MASTER STATE
Jul 2 16:06:48 packer-vmware-iso Keepalived_vrrp[3873]: VRRP_Instance(VI_1) Entering MASTER STATE
Jul 2 16:08:53 packer-vmware-iso Keepalived_vrrp[3873]: VRRP_Instance(VI_1) Received lower prio advert, forcing new election

```

图 11-2

在没有绑定浮动 IP 地址的节点上，会看到如图 11-3 所示输出。


```

Jul 2 16:08:52 packer-vmware-iso Keepalived[3886]: Starting VRRP child process, pid=3890
Jul 2 16:08:53 packer-vmware-iso Keepalived_vrrp[3890]: VRRP_Script(chk_haproxy) succeeded
Jul 2 16:08:53 packer-vmware-iso Keepalived_vrrp[3890]: VRRP_Instance(VI_1) Transition to MASTER STATE
Jul 2 16:08:53 packer-vmware-iso Keepalived_vrrp[3890]: VRRP_Instance(VI_1) Received higher prio advert
Jul 2 16:08:53 packer-vmware-iso Keepalived_vrrp[3890]: VRRP_Instance(VI_1) Entering BACKUP STATE

```

图 11-3

使用浮动 IP 地址配置 OpenStack 后端

两个 HA Proxy 服务器均运行相同的 HA Proxy 设置以及 keepalived，可使用虚拟 `virtual_ipaddress` 地址（浮动 IP 地址）作为配置文件中连接数据库的地址。在 OpenStack 中，找到指向数据库的每个配置文件，然后修改以下配置为使用浮动 IP 地址 172.16.0.251。

1. 首先，必须确保新的 Galera 集群具备 OpenStack 环境所需的用户名和密码。在本书配套的测试 `vagrant` 环境中，我们将数据库用户名配置为与服务的名称相同，如 `neutron`，密码均设置为 `openstack`。执行如下命令，创建所有用户和密码。

```

USERS="nova
neutron
keystone
glance
cinder
heat"

HAPROXIES="172.16.0.248
172.16.0.249"

for U in ${USERS}
do
    for H in ${HAPROXIES}
    do
        mysql -u root -h localhost -e "GRANT ALL ON *.* to
${U}@\"${H}\" IDENTIFIED BY \"openstack\";"
    done
done

```



建议在生产环境中使用更强、更随机的密码。

2. 现在可使用上述信息，替换掉 OpenStack 配置文件中使用的 SQL 连接设置，示例如下。

```

# Nova
# /etc/nova/nova.conf
sql_connection = mysql://nova:openstack@172.16.0.251/nova

# Keystone
# /etc/keystone/keystone.conf
connection =

```

```
mysql://keystone:openstack@172.16.0.251/keystone

# Glance
# /etc/glance/glance-registry.conf
connection = mysql://glance:openstack@172.16.0.251/glance

# Neutron
# /etc/neutron/neutron.conf
[DATABASE]
connection = mysql://neutron:openstack@172.16.0.251/neutron

# Cinder
# /etc/cinder/cinder.conf
connection = mysql://cinder:openstack@172.16.0.251/cinder
```

工作原理

我们安装并配置了 keepalived，这个服务可以让我们让多个 HA Proxy 服务器共享一个浮动 IP 地址。在出现故障时，另一个 HA Proxy 服务器会接管剩余的服务器。

我们通过编辑 /etc/keepalived/keepalived.conf 文件，来配置 keepalived 服务。两个节点上的配置类似，只有一个区别，即需要区分主节点和从节点。

我们选择第一个 HA Proxy 服务器作为主节点上（可以是任何命名实例），可通过如下代码实现。

```
vrrp_instance VI_1 {
    interface eth1 # interface to monitor
    state MASTER
    virtual_router_id 51 # Assign one ID for this route
    priority 101 # 101 on master, 100 on backup
    virtual_ipaddress {
        172.16.0.251 # the virtual IP
    }
}
```

在从节点上，具体的代码为。

```
vrrp_instance VI_1 {
    interface eth1 # interface to monitor
    state BACKUP
    virtual_router_id 51 # Assign one ID for this route
    priority 100 # 101 on master, 100 on backup
    virtual_ipaddress {
        172.16.0.251 # the virtual IP
    }
}
```

在本例中，我们使用的 IP 地址 172.16.0.251 可以在实例之间进行浮动。在上面的代码块中 virtual_ipaddress 部分有设置。

在两台服务器上启动 keepalived 之后，主节点获得了 172.16.0.251 这个 IP 地址。如果将该服务器关机，或者意外故障，另一台 HA Proxy 服务器便会继承该 IP 地址。这样

可以实现 HA Proxy 服务器的高可用。

这样，就可以确保新数据库会配置好所有相关的用户名和密码，然后再替换掉之前非 HA Proxy 配置下的链接。

11.5 使用 Corosync 安装并配置 Pacemaker

OpenStack 是一个高伸缩的环境，可以避免单点故障，但是有时需要在自己构建的环境下才能实现该特性。例如，Keystone 是整个 OpenStack 环境的基础、核心服务，因此必须构建多个 keystone 实例。Glance 是另一个运行 OpenStack 环境的关键服务。通过配置多台实例运行这些服务，并使用 Pacemaker 和 Corosync 进行控制可获得更大的弹性，来应对服务节点的故障。使用 Pacemaker 和 Corosync 是保障 OpenStack 服务高可用的一种方法。本节将介绍一些部署方式，以及如何在环境中的其他部分使用 Pacemaker 和 Corosync。

准备工作

本节中，我们假设有两个控制节点运行 Glance 和 Keystone 服务。本书前两章介绍了如何安装 Keystone 和 Glance。

第一个控制节点 controller1 的管理地址为 192.168.100.221，第二个节点 controller2 的管理地址为 192.168.100.222。



访问 <https://github.com/OpenStackCookbook/Controller-Corosync.git>，查看本节所说的双节点 OpenStack 控制节点实例。

操作步骤

执行如下步骤，在将要运行 Keystone、Glance 等 OpenStack 服务的两台服务器上安装 Pacemaker 和 Corosync。

设置第一个节点：controller1

1. 在 OpenStack 环境中安装 Keystone 和 Glance 服务，并配置可供通信的地址之后，执行如下步骤安装 Pacemaker 和 Corosync。

```
sudo apt-get update
sudo apt-get install pacemaker corosync
```

2. 两个节点需要知道各自的地址和主机名，因此在 /etc/hosts 文件中写入相关信息，避免 DNS 查询。


```
192.168.100.221 controller1.book controller1
192.168.100.222 controller2.book controller2
```

3. 编辑/etc/corosync/corosync.conf 文件, 使得接口部分的设置如下所示。

```
interface {
    # The following values need to be set based on your
    environment
    ringnumber: 0
    bindnetaddr: 192.168.100.0
    mcastaddr: 226.94.1.1
    mcastport: 5405
}
```



Corosync 使用的是组播 (multicast)。请确保具体设置不会与网络上其他启用了组播的服务相冲突。

4. 默认情况下, corosync 服务不会设置为启动。如需启动, 编辑/etc/default/corosync 服务, 并设置 START=yes, 具体如下。

```
sudo sed -i 's/^START=no/START=yes/g' /etc/default/corosync
```

5. 现在需要生产一个验证密钥, 确保主机间的通信安全。

```
sudo corosync-keygen
```

6. 程序会询问你是否使用键盘生成一个随机熵。如果是通过 SSH 而非控制台连接, 将无法使用键盘生成。如需远程完成这一步, 启动一个新的 SSH 会话。在新的会话中, 在 corosync 命令等待输入的同时, 运行如下命令。

```
while /bin/true
do
    dd if=/dev/urandom of=/tmp/100 bs=1024 count=100000
    for i in {1..10}
    do
        cp /tmp/100 /tmp/tmp_$i_$RANDOM
    done
    rm -f /tmp/tmp_* /tmp/100
done
```

7. corosync-keygen 命令运行结束之后, 会生成一个 authkey 文件, 之后按下 Ctrl+C 组合键即可退出命令循环。

设置第二个节点: controller2

现在需要在第二个节点 controller2 上安装 Pacemaker 和 Corosync。

1. 执行如下步骤, 安装 pacemaker 和 corosync 软件包。

```
sudo apt-get update
```

```
sudo apt-get install pacemaker corosync
```

2. 还要确保/etc/hosts 文件中的内容与第一个节点的文件相同。

```
192.168.100.221 controller1.book controller1
192.168.100.222 controller2.book controller2
```

3. 默认情况下, corosync 服务不会设置为启动。如需启动, 编辑/etc/default/corosync 服务, 并设置 START=yes, 具体如下。

```
sudo sed -i 's/^START=no/START=yes/g' /etc/default/corosync
```

配置第一个节点: controller1

修改/etc/corosync/corosync.conf 文件, 在生成/etc/corosync/authkey 文件之后, 将它们拷贝到集群的其他节点上。

```
scp /etc/corosync/corosync.conf /etc/corosync/authkey
openstack@ 192.168.100.222:
```

配置第二个节点: controller2

现在可将从第一个节点中拷贝的 corosync.conf 和 authkey 文件, 放置到/etc/corosync 目录下。

```
sudo mv corosync.conf authkey /etc/corosync
```

启动 Pacemaker 和 Corosync 服务

1. 现在可启动这两个服务。在两个节点上, 运行如下命令。

```
sudo service pacemaker start
sudo service corosync start
```

2. 可使用 crm_mon 命令查询集群的状态, 检查服务是否正常启动, 集群状态是否正常。

```
sudo crm_mon -l
```

上述命令的输出如图 11-4 所示, 其中的重要信息包括配置的节点数量, 预期节点数量以及在线的节点情况。

```
Last updated: Sun Mar 29 12:32:45 2015
Last change: Sun Mar 29 12:32:14 2015 via crmd on controller1
Stack: corosync
Current DC: controller1 (739246301) - partition with quorum
Version: 1.1.10-42f2063
2 Nodes configured
0 Resources configured

Online: [ controller1 controller2 ]
```

图 11-4

3. 可使用 `crm_verify` 命令, 验证配置情况。

```
sudo crm_verify -L -V
```

4. 上述命令会报错, 提示 STONISH (Shoot The Other Node in The Head 的简称)。STONISH 用于在至少配置了三个节点时维持法定节点数量。在双节点集群中并不做此要求。由于我们只配置一个双节点集群, 因此禁用 stonish。

```
sudo crm configure property stonith-enabled=false
```

5. 再次通过 `crm_verify` 命令验证集群则会报错。

```
sudo crm_verify -L
```

6. 另外, 由于这只是一个双节点集群, 我们使用如下命令禁用 quorum 策略。

```
sudo crm configure property no-quorum-policy=ignore
```

7. 在第一个节点 `controller1` 上, 可以配置服务, 并设置一个两台服务器共享的浮动 IP 地址。在下面的命令中, 我们选择 192.168.100.253 作为浮动 IP 地址, 每隔 5 秒监控一次。为此, 我们再次使用 `crm` 命令配置该 IP 地址, 具体命令如下。

```
sudo crm configure primitive FloatingIP \
    ocf::heartbeat:IPaddr2 params ip=192.168.100.253 \
    cidr_netmask=32 op monitor interval=5s
```

8. 使用 `crm_mon` 命令查看集群的状态后, 可以看到上面的浮动 IP 地址已经指派给了 `controller1` 主机。

```
sudo crm_mon -l
```

上述命令的输出如图 11-5 所示, 目前只配置了一个资源。

```
Last updated: Sun Mar 29 15:18:06 2015
Last change: Sun Mar 29 15:13:23 2015 via cibadmin on controller2
Stack: corosync
Current DC: controller1 (1084777693) - partition with quorum
Version: 1.1.10-42f2063
2 Nodes configured
1 Resources configured

Online: [ controller1 controller2 ]

FloatingIP      (ocf::heartbeat:IPaddr2):      Started controller1
```

图 11-5

9. 现在可使用 192.168.100.253 这个浮动 IP 地址连接到第一个节点。在节点关机之后, 如果 5 秒后第一个节点没有响应, 该地址将被赋给第二个节点。可从任意一个控制节点上执行如下命令, 测试该浮动 IP。

```
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
```



```
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.253:5000/v2.0/
```

```
keystone --insecure endpoint-list
```

上述命令的输出如图 11-6 所示。

id	name	enabled
ee59bfbf11924c90a535a2eb7a9390e1	cookbook	True
cd849efa42b7471d819d6c196060b8cb	service	True

图 11-6



注意，在命令行中使用了 `-insecure` 标志，这是因为我们在两个控制节点上用的是独立生成的自签名证书。在生产环境下，不要设置该标志。

工作原理

OpenStack 服务实现高可用是一个非常复杂的话题，实现方式有多种。Pacemaker 和 Corosync 是其中很好的方案，可以配置一个浮动 IP 地址，指定给集群，并使用 Corosync 添加到合适的节点上；还可以使用代理 agents 来控制服务，集群管理器可以按需启动和停止服务，为最终用户提供高可用的体验。

我们在两个节点上都安装了 Keystone 和 Glance（每个服务都配置了一个对应的远程数据库后端，如 MySQL 和 Galera），使用共享文件系统或云存储方案来获取镜像。这可以让我们方便地使用 Pacemaker 来配置和监控服务。如果在主节点上没有所需的服务，Pacemaker 可在备节点上启动这些服务。

11.6 使用 Pacemaker 和 Corosync 配置 OpenStack 服务

本节将使用运行 Glance 和 Keystone 服务的两个节点，它们由 Pacemaker 通过 Corosync 控制，设置为主/备模式，可允许单个节点出错。在生产环境下，建议集群至少包含三个节点，确保一个节点故障后仍能保持弹性和一致性。

准备工作

本节中，我们假设读者已经按照上节的步骤配置好了两个控制节点 `controller1` 和 `controller2`，并具备 Corosync 提供的浮动 IP 地址 `172.16.0.253`。

操作步骤

执行如下步骤，提升 OpenStack 服务的弹性。

1. 由于 controller1 上运行了 Keystone，我们应该从一个可访问 OpenStack 环境的客户机上，使用其 IP 地址（172.16.0.221）和浮动 IP 地址（172.16.0.253）查询 Keystone。

```
# Assigned IP (192.168.100.221)
export OS_TENANT_NAME=cookbook
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=https://192.168.100.221:5000/v2.0/
export OS_KEY=/vagrant/cakey-controller1.pem
export OS_CACERT=/vagrant/ca-controller1.pem
```

```
keystone user-list
```

```
# FloatingIP (Keepalived and HA Proxy)
export OS_AUTH_URL=https://172.16.0.253:5000/v2.0/
```

```
keystone user-list
```

2. 从第一个节点上复制/etc/keystone/keystone.conf 文件到第二个节点，然后重启 keystone 服务。不需要做其他工作，因为第一个节点上安装完之后，数据库中就已经写入了相应的端点和用户。重启服务，重新连接到数据库，具体如下。

```
sudo stop keystone
sudo start keystone
```

3. 现在可以在第二个节点的 IP 地址上查询其 keystone 服务。

```
# Second Node
export OS_AUTH_URL=http://172.16.0.112:5000/v2.0/
keystone user-list
```

两个节点上的 Glance 配置浮动 IP

为了让 Glance 能在多个节点上运行，必须配置使用一个共享的存储后端（如 Swift），并连接到数据库后端（如 MySQL）。在第一个主机上，按照 2.1 节的步骤安装并配置 Glance。之后，执行如下步骤。

1. 在第二个节点上，安装 Glance 所需的软件包，并连接到 MySQL 和 Swift，具体命令如下。

```
sudo apt-get install glance python-swift
```

2. 复制/etc/glance 中的配置文件到第二个节点，在两个节点上启动 glance-api 和 glance-registry 服务。

```
sudo start glance-api
sudo start glance-registry
```

3. 现在可使用任意一个 Glance 服务器查看镜像, 以及赋给第一个节点的浮动 IP 地址, 具体命令如下。

```
# First node
glance -I admin -K openstack -T cookbook -N
    http://172.16.0.111:5000/v2.0 index
# Second node
glance -I admin -K openstack -T cookbook -N
    http://172.16.0.112:5000/v2.0 index
# FloatingIP
glance -I admin -K openstack -T cookbook -N
    http://172.16.0.253:5000/v2.0 index
```

配置 Pacemaker 使用 Glance 和 Keystone

两个节点上均运行 Keystone 和 Glance 之后, 可以配置 Pacemaker 来控制这两个服务, 以便确保其中一个节点出错时 Keystone 和 Glance 在正常的节点上运行, 具体步骤如下。

1. 首先禁用控制 Keystone 和 Glance 的 upstart 作业, 然后创建在两个节点上对应的 upstart 覆盖文件。创建/etc/init/keystone.override、/etc/init/glance-api.override 和/etc/init/glance-registry.override 文件, 只填写关键词 manual。

2. 现在获取 OCF (全称 Open Cluster Format) 资源代理 agent, 它们是能够控制 Keystone 和 Glance 服务的 shell 脚本或代码。为此, 运行如下命令。

```
wget https://raw.githubusercontent.com/madkiss/keystone/ha/tools/ocf/keystone
wget https://raw.githubusercontent.com/madkiss/glance/ha/tools/ocf/glance-api
wget https://raw.githubusercontent.com/madkiss/glance/ha/tools/ocf/glance-registry
sudo mkdir -p /usr/lib/ocf/resource.d/openstack
sudo cp keystone glance-api glance-registry \
    /usr/lib/ocf/resource.d/openstack
sudo chmod 755 /usr/lib/ocf/resource.d/openstack/*
```

3. 现在应该能查询可用的 OCF 代理, 目前会返回三个。

```
sudo crm ra list ocf openstack
```

4. 现在可以配置 Pacemaker 使用这些代理, 控制 Keystone 服务, 执行如下命令。

```
sudo crm cib new conf-keystone
sudo crm configure property stonith-enabled=false
sudo crm configure property no-quorum-policy=ignore
sudo crm configure primitive p_keystone
ocf:openstack:keystone \
    params config="/etc/keystone/keystone.conf" \
    os_auth_url="http://localhost:5000/v2.0/" \
    os_password="openstack" \
    os_tenant_name="cookbook" \
    os_username="admin" \
    user="keystone" \
    client_binary="/usr/bin/keystone" \
    op monitor interval="5s" timeout="5s"
sudo crm cib use live
```



```
sudo crm cib commit conf-keystone
```

5. 然后针对 Glance 服务, 执行如下命令。

```
sudo crm cib new conf-glance-api
sudo crm configure property stonith-enabled=false
sudo crm configure property no-quorum-policy=ignore
sudo crm configure primitive p_glance_api
ocf:openstack:glance-api \
    params config="/etc/glance/glance-api.conf" \
    os_auth_url="http://localhost:5000/v2.0/" \
    os_password="openstack" \
    os_tenant_name="cookbook" \
    os_username="admin" \
    user="glance" \
    client_binary="/usr/bin/glance" \
    op monitor interval="5s" timeout="5s"
sudo crm cib use live
sudo crm cib commit conf-glance-api
```

```
sudo crm cib new conf-glance-registry
sudo crm configure property stonith-enabled=false
sudo crm configure property no-quorum-policy=ignore
sudo crm configure primitive p_glance_registry \
ocf:openstack:glance-registry \
    params config="/etc/glance/glance-registry.conf" \
    os_auth_url="http://localhost:5000/v2.0/" \
    os_password="openstack" \
    os_tenant_name="cookbook" \
    os_username="admin" \
    user="glance" \
    op monitor interval="5s" timeout="5s"
sudo crm cib use live
sudo crm cib commit conf-glance-registry
```

6. 执行如下命令, 验证是否成功配置 Pacemaker 来控制服务:

```
sudo crm_mon -1
```

上述命令的输出如图 11-7 所示。

```
Last updated: Sat Aug 24 22:55:25 2015
Last change: Tue Aug 24 21:06:10 2015 via crmd on
controller1
Stack: openais
Current DC: controller1 - partition with quorum
Version: 1.1.6-9971ebba4494012a93c03b40a2c58ec0eb60f50c
2 Nodes configured, 2 expected votes
4 Resources configured.
=====
Online: [ controller1 controller2 ]

FloatingIP (ocf::heartbeat:IPaddr2): Started controller1
p_keystone (ocf::openstack:keystone):
    Started controller1
p_glance_api (ocf::openstack:glance_api):
    Started controller1
p_glance_registry (ocf::openstack:glance_registry):
    Started controller1
```

图 11-7

如果出现类似如下错误:



Failed actions:

```
p_keystone_monitor_0 (node=controller2, call=3,
rc=5, status=complete): not installed
```

执行如下命令清除状态, 并再次查看:

```
sudo crm_resource -P
sudo crm_mon -l
```

7. 现在配置客户机, 让其使用浮动 IP 地址 (172.16.0.253) 访问 Glance 和 Keystone 服务。配置好之后, 可以将第一个节点上的网络接口卸除, 但是仍可以通过浮动 IP 地址访问 Keystone 和 Glance 服务。

现在, Keystone 和 Glance 服务在两个不同的节点上运行, 即使其中一个故障, 服务仍然是可用的。

工作原理

Pacemaker 的配置主要通过 crm 工具完成, 支持通过脚本的方式进行。如果直接调用 crm, 会启动一个交互式 shell, 可用来编辑、添加和移除服务, 以及查询集群的状态。这是一个非常强大的工具, 可控制同样十分强大的集群管理器。

两个节点均运行 Keystone 和 Glance, 并且配置好 Pacemaker 和 Corosync 后, 可通过 Corosync 提供的浮动 IP 访问, 之后我们配置 Pacemaker 来控制 Keystone 和 Glance 服务。这是通过专门为此设计的 Open Cluster Framework (OCF) 实现的。OCF 代理要求的一些参数我们都很熟悉, 与客户机访问服务所需的用户名、密码、租户和端点 URL 相同。

代理设置了 5 秒的超时, 5 秒后浮动 IP 地址就会切换到另一个节点。

这样配置完成之后, 就实现了 Keystone 和 Glance 的主/备配置, 具体如图 11-8 所示。

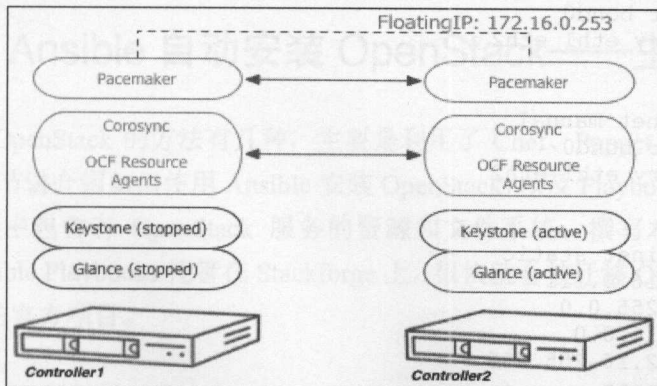


图 11-8

11.7 绑定多个网卡实现高冗余

将多个服务分散在多台机器上运行，并实现相应的高可用手段，可确保环境具备高度容灾特性。但如果物理网络出现故障，而不是服务本身，那么该服务处理的流量将无法流入流出，也会造成服务不可用。添加网卡绑定（NIC bonding）功能，有助于缓解这些问题，通过多个路由器和交换器确保网络畅通。

准备工作

网卡绑定要求系统管理员与负责交换机的网络管理员进行密切协作。实现网卡绑定的方法很多，本节介绍的方式是主备模式，指的是流量正常情况下通过一个交换机流入流出，另一个绑定的网卡在非必要情况下不做处理。

操作步骤

在 Ubuntu 14.04 上进行网卡绑定前，要再安装一个软件包来支持该功能，具体操作步骤如下。

1. 安装 ifenslave 软件包。

```
sudo apt-get update
sudo apt-get install ifenslave
```

2. 安装好 ifenslave 软件包后，通过正常方式在 Ubuntu 中配置网络，但是加入绑定多个网卡所要求的元素。编辑/etc/network/interfaces 文件，加入如下内容（配置主备模式），这里我们将 eth1 和 eth2 绑定在一起，得到一个地址为 172.16.0.111 的 bond0 接口。

```
auto eth1
iface eth1 inet manual
    bond-master bond0
    bond-primary eth1 eth2

auto eth2
iface eth2 inet manual
    bond-master bond0
    bond-primary eth1 eth2

auto bond0
iface bond0 inet static
address 172.16.0.111
netmask 255.255.0.0
network 172.16.0.0
broadcast 172.16.255.255
bond-slaves none
bond-mode 1
bond-miimon 100
```


3. 为了确保使用的是正确的绑定模式，在 `/etc/modprobe.d/bonding.conf` 文件中加入如下内容，设置为主备绑定模式（`mode=1`），每个 100 毫秒监测一次。

```
alias bond0 bonding
options bonding mode=1 miimon=100
```

4. 现在重启网络服务，这样会以设置的 IP 地址启动绑定后的网卡。

```
sudo service networking restart
```

工作原理

在 Ubuntu 中绑定多个网卡来应对交换机故障，是比较直接的操作，主要协调交换机的设置和配置实现。配置好连接到不同交换机的通路，每个网卡连接到不同的交换机后，就可实现网络层级故障（如交换机故障）的高容灾。

为此，我们在 Ubuntu 下的 `/etc/network/interfaces` 文件中配置了绑定，但是指定了哪些网卡组成绑定后的接口。每个配置的绑定网卡至少连接一对独立的接口，然后在使用 IP 地址、`netmask` 等常用工具配置绑定后的接口 `bond0`。我们还特别注明了一些选项，告知 Ubuntu 这是特定模式下的绑定网卡。

为了确保绑定模块随内核加载后被赋予正确的模式，我们在 `/etc/modprobe.d/bonding.conf` 文件中进行了配置。在网卡与绑定模块一起加载时，就得到了一个能够应对交换机故障的服务器。

更多参考

更多有关 Ubuntu Linux 所支持的绑定模式，请参考：

<https://help.ubuntu.com/community/LinkAggregation>。

11.8 使用 Ansible 自动安装 OpenStack——主机配置

自动化安装 OpenStack 的方法有几种，主要是利用了 Chef、Puppet 和 Ansible 等配置管理工具实现。本节将介绍如何使用 Ansible 安装 OpenStack，以及 Playbooks 如何利用 LXC 容器。LXC 容器中包含有 OpenStack 服务的资源和文件系统。撰写本书时，用于安装 OpenStack 的 Ansible Playbooks 托管在 Stackforge 上，很快就会被迁移 OpenStack 的 Github 项目，作为单独的官方项目。

准备工作

本节将要使用的环境由七个物理服务器组成。

- ❑ 控制节点组成一个节点集群，运行 OpenStack API 服务，包括 Glance、Keystone、Horizon，以及 MariaDB 和 RabbitMQ。
- ❑ 一个存储节点用于 Cinder LVM 卷。
- ❑ 两个（或多个）计算节点。在这种架构下，可以横向扩容以满足环境的计算需求，或者将计算服务分散在更大的节点集群中，按要求扩容。
- ❑ 一个 HA Proxy 节点将被用于环境安装，同时为 OpenStack 服务提供负载均衡功能。



在生产环境中，强烈建议使用一对物理负载均衡器代替 HA Proxy。

- ❑ 所有机器都需要全新安装 Ubuntu 14.04 LTS 系统。
- ❑ 所有机器都需能够访问互联网。

每个服务器至少要安装两个网卡，并利用 VLAN（一共要创建 4 个独立网络）。在生产环境中，应该至少有 4 个网卡，这样可以创建两组绑定网卡，并连接到不同的 HA 交换机，实现系统弹性。

为了更好地理解环境的网络设置，请参考图 11-9。

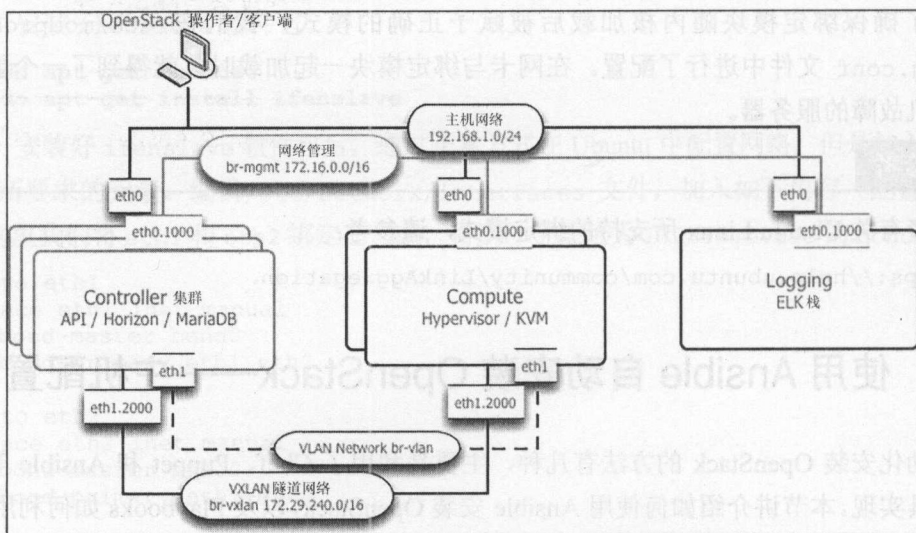


图 11-9

使用 Ansible 安装 OpenStack 时，会使用如下网络。

- ❑ eth0: 用于访问主机本身（未标记的 VLAN）。这个接口的 IP 地址位于主机的子网上，将用于存储流量。也可使用 br-storage 桥接，专门处理存储流量。本节

中没有使用该桥接。

- ❑ eth0.1000: 一个 VLAN (标签 1000) 接口, 容器桥接 (br-mgmt) 在该接口上创建。eth0.1000 接口不会被直接指定 IP 地址; 而是指定给桥接 (br-mgmt)。
 - br-mgmt: 该桥接连接到 eth0.1000, 是专门用于处理容器网络流量的网络接口。该网络将处理 OpenStack 服务之间的通信, 如 Glance 要求访问 Keystone。br-mgmt 桥接的 IP 地址位于管理网络 (也叫容器网络), 这样主机就可以访问容器。
- ❑ eth1: 所有基于 VLAN 的 Neutron 流量都会经过这个网络接口。控制节点和计算节点要求配置该接口。存储节点和 HA Proxy 节点不需要配置该网络。
 - br-vlan: 是连接到 eth1 的桥接。br-vlan 和 eth1 都没有 IP 地址, 因为 OpenStack Neutron 会在创建 VLAN 类型的网络时即时进行控制。
- ❑ eth1.2000: 会在该 VLAN (标签 2000) 接口上创建一个 VXLAN 网络。OpenStack Neutron 能够创建 VXLAN 类型的私有租户网络。在该接口上创建的网络如下。
 - br-vxlan: 该桥接包含了 eth1.2000, 用于传递 VXLAN 隧道网络中生成的数据。在本书设计的 OpenStack 环境中, 该网络允许用户创建 VXLAN 类型的 Neutron 网络, 叠加在该网络之上。桥接的 IP 地址来自隧道网络。

操作步骤

第一阶段是确保本节所说的七个主机都完成正确配置, 可使用 Ansible Playbook 安装; 为此, 执行如下步骤。

1. 配置七个主机上的网络。编辑 /etc/network/interfaces 文件, 加入以下内容 (生产环境中可使用绑定网卡, 请按实际的网络需求进行编辑)。

```
# Host Interface
auto eth0
iface eth0
inet static
address 192.168.1.101
netmask 255.255.255.0
gateway 192.168.1.1
dns-nameservers 192.168.1.1

# Neutron Interface, no IP assigned
auto eth1
iface eth1
inet manual

# Container management VLAN interface
iface eth0.1000
inet manual
```



```

vlan-raw-device eth0

# OpenStack VXLAN (tunnel/overlay) VLAN interface
iface eth1.2000
    inet manual
vlan-raw-device eth1

```

2. 继续编辑/etc/network/interfaces 文件，加入对应的桥接信息。

```

# Bridge for Container network
auto br-mgmt
    iface br-mgmt inet static
        bridge_stp off
        bridge_waitport 0
        bridge_fd 0
    # Bridge port references tagged interface
    bridge_ports eth1.1000
    address 172.16.0.101
    netmask 255.255.0.0
    dns-nameservers 192.168.1.1

# Bridge for vlan network
auto br-vlan
    iface br-vlan inet manual
        bridge_stp off
        bridge_waitport 0
        bridge_fd 0
    # Notice this bridge port is an Untagged interface
    bridge_ports eth1

# Bridge for vxlan network
auto br-vxlan
    iface br-vxlan inet static
        bridge_stp off
        bridge_waitport 0
        bridge_fd 0
    # Bridge port references tagged interface
    bridge_ports eth1.2000
    address 172.29.240.101
    netmask 255.255.252.0
    dns-nameservers 192.168.1.1

```

3. 现在可重启网络服务，这将以设置的 IP 地址启动主机的网络接口和桥接。

```
sudo service networking restart
```

4. 在要安装 OpenStack 的每个主机上，重复第一步至第三步，调整对应的 IP 地址。

5. 确保环境中的七个节点均可在创建的网络上访问。可使用 fping 命令实现。

```

# host network (eth0)
fping -g 192.168.1.101 192.168.1.107

# container network (br-mgmt)
fping -g 172.16.0.101 172.16.0.107

```

```
# For Computers and Controllers Only
# tunnel network (br-vxlan)
fping -g 172.29.240.101 172.29.240.105
```

工作原理

正确配置网络功能很重要。如果安装完 OpenStack 环境后再调整网络，可能就会比较棘手。

我们用了两个物理接口（如果使用绑定，一共就是 4 个，但视作 2 个），配置了对应的 VLAN，并放置到了特定的桥接里。Ansible Playbook 的配置中直接引用了桥接 br-mgmt、br-vxlan 和 br-vlan，因此不要修改这些名称。

eth0 上的主机网络，包含了 LAN 的默认网关，在安装 OpenStack 过程中将使用该网络从因特网上下载所需的软件包。

我们在 eth0 上创建了一个标签为 eth0.1000 的 VLAN，用于处理容器间流量。Ansible Playbook 会在 LXC 容器里安装 OpenStack 服务，因此这些容器必须能够相互通信。该网络不可路由，只用于容器间通信。这个 VLAN 网络会被放入桥接 br-mgmt，赋予一个管理（容器）网络上的 IP 地址，这样后两节创建的主机就可以与容器进行通信。

第二个接口（或绑定接口）负责处理 Neutron 流量，因此只有控制节点和计算节点需要该接口。由于我们配置环境为支持 VLAN Neutron 租户网络和 VXLAN 租户网络，首先创建一个 VLAN 网络 eth1.2000，然后放入桥接 br-vxlan。由于它是处理 VXLAN 流量的，我们为其制定一个 IP。现在可在该网络上创建隧道。该网络没有任何相关联的路由器。然后创建一个 br-vlan 桥接，将 eth1 放入桥接，这是因为我们最终要创建 VLAN 类型的 Neutron 租户网络，Neutron 会给该接口打标签。

11.9 使用 Ansible 自动安装 OpenStack——Playbook 配置

正确配置好主机和网络接口之后，可以开始编辑 Ansible Playbook 运行时使用的配置文件。本节中，我们使用 Git 来检查 OSAD Playbooks，即由 Rackspace 开发、用来帮助客户部署 OpenStack 的 Playbook 文件。我们将使用本书撰写时的最新版：Git 标签为 11.0.3 的 Kilo 发行版（Kilo 指的是字母 K，是字母表中的第 11 个字母）。

我们将配置的环境如图 11-10 所示。

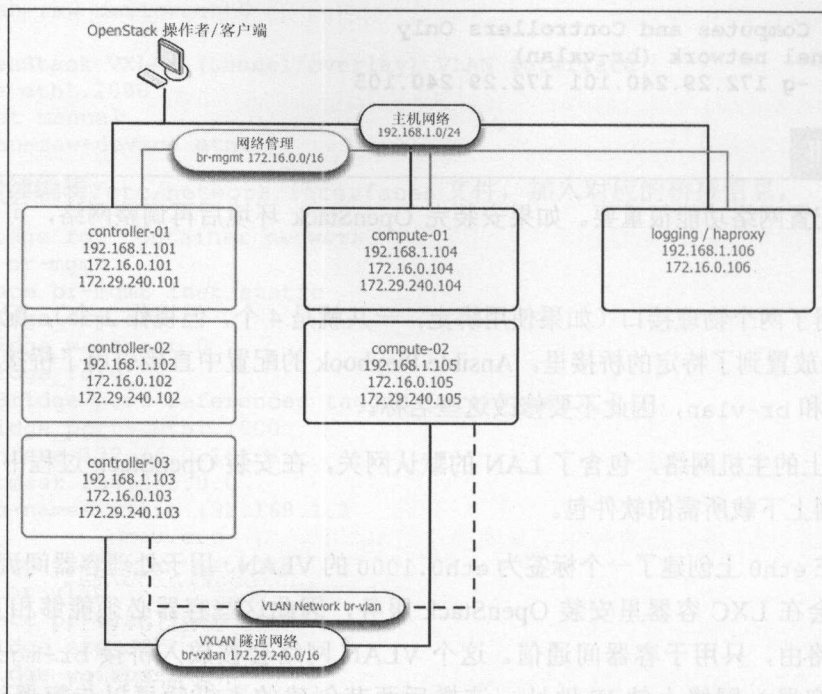


图 11-10

准备工作

按照 11.8 节所述操作，完成主机的配置，并且要求所有配置的网络正常运行。

环境包含三个控制节点，一个存储节点，一个 HA Proxy 节点和两个计算节点。找到 HA Proxy 服务器，并以 root 用户身份登录。为了方便，使用该服务器来安装 OpenStack。

操作步骤

本节中，我们将配置 Playbook 使用的 YAML 配置文件，包括 `openstack_user_config.yml`、`user_variables.yml` 和 `user_secrets.yml` 三个文件。它们共同描述了整个安装过程，包括从哪个服务器运行何种 OpenStack 功能，到具体的密码和启用的特性。

1. 首先从 Github 获取 Ansible Playbook，放在 `/opt/os-ansible-deployment` 目录下。使用如下命令实现。

```
cd /opt
git clone -b 11.0.3 \
```

```
https://github.com/stackforge/os-ansible-deployment.git
```



2. 然后配置安装过程。首先从克隆的 GitHub 仓库中复制实例和空的配置文件到 `/etc/openstack_deploy` 目录，具体命令如下。

```
cp -R /opt/os-ansible/etc/openstack_deploy /etc
```

3. 第一个配置的文件时位于 `/etc/openstack_deploy` 目录下的大文件 `openstack_user_config.yml`，它描述了物理环境的信息，包括网络地址段、使用的网络接口以及运行每个服务的节点。第一部分指的是环境中使用的 CIDR。

```
---
cidr_networks:
  management: 172.16.0.0/16
  tunnel: 172.29.240.0/22

used_ips:
  - 172.16.0.101,172.16.0.107
  - 172.29.240.101,172.29.240.107
```

 该文件的网络下载地址为：https://github.com/OpenStackCookbook/OpenStackCookbook/blob/master/ansible-openstack/openstack_user_config.yml。

4. 这个文件中还有 `global_override` 部分，描述了负载均衡的 VIP 地址，网络桥接以及 Neutron 网络的细节。这部分的设置更长。注意，我们在提前描述具体的安装细节。这里，我们设置了一个负载均衡使用的 VIP，但实际尚未存在。我们将通过 HA Proxy（下一节安装）来使用这些地址。

```
global_overrides:
  internal_lb_vip_address: 172.16.0.107
  external_lb_vip_address: 192.168.1.107
  lb_name: haproxy
  tunnel_bridge: "br-vxlan"
  management_bridge: "br-mgmt"
  provider_networks:
    - network:
        group_binds:
          - all_containers
          - hosts
        type: "raw"
        container_bridge: "br-mgmt"
        container_interface: "eth1"
        container_type: "veth"
        ip_from_q: "management"
        is_container_address: true
        is_ssh_address: true
    - network:
        group_binds:
          - neutron_linuxbridge_agent
        container_bridge: "br-vxlan"
```

```

    container_type: "veth"
    container_interface: "eth10"
    ip_from_q: "tunnel"
    type: "vxlan"
    range: "1:1000"
    net_name: "vxlan"
- network:
  group_binds:
    - neutron_linuxbridge_agent
  container_bridge: "br-vlan"
  container_type: "veth"
  container_interface: "eth11"
  type: "vlan"
  range: "1:1"
  net_name: "vlan"
- network:
  group_binds:
    - neutron_linuxbridge_agent
  container_bridge: "br-vlan"
  container_type: "veth"
  container_interface: "eth12"
  host_bind_override: "eth12"
  type: "flat"
  net_name: "flat"

```

5. 这部分完成后，就到了描述由哪些服务器组成 OpenStack 环境的部分。还是在同一个文件中，添加如下内容。每个部分包括 3 个服务器，我们将其分配为控制节点。这部分是关于 MariaDB 和 RabbitMQ 等共享服务的设置。

```

# Shared infrastructure parts
shared-infra_hosts:
  controller-01:
    ip: 172.16.0.101
  controller-02:
    ip: 172.16.0.102
  controller-03:
    ip: 172.16.0.103

```

6. 这部分用于安装 OpenStack 计算服务，如 Nova API:

```

# OpenStack infrastructure parts
os-infra_hosts:
  controller-01:
    ip: 172.16.0.101
  controller-02:
    ip: 172.16.0.102
  controller-03:
    ip: 172.16.0.103

```

7. storage-infra 部分描述 Cinder 存储 API 相关设置。

```

# OpenStack Storage infrastructure parts
storage-infra_hosts:
  controller-01:

```

```

ip: 172.16.0.101
controller-02:
ip: 172.16.0.102
controller-03:
ip: 172.16.0.103

```

8. 下面这部分介绍如何访问 Keystone API。

```

# Keystone Identity infrastructure parts
identity_hosts:
  controller-01:
    ip: 172.16.0.101
  controller-02:
    ip: 172.16.0.102
  controller-03:
    ip: 172.16.0.103

```

9. 下面，描述用作计算节点的服务器。在同一个文件中，添加如下细节，描述计算主机。

```

# Compute Hosts
compute_hosts:
  compute-01:
    ip: 172.16.0.104
  compute-02:
    ip: 172.16.0.105

```

10. 接下来，配置 Cinder 存储节点。添加具体配置方式（如使用 NetAPP 或 LVM 等后端）。

```

storage_hosts:
  storage:
    ip: 172.16.0.106
    container_vars:
      cinder_backends:
        limit_container_types: cinder_volume
      lvm:
        volume_group: cinder-volumes
        volume_driver:
cinder.volume.drivers.lvm.LVMISCSIDriver
        volume_backend_name: LVM_iSCSI

```

11. Playbook 中，还说明在多个基础设施节点上安装 Neutron 服务。我们让 Ansible 在如下地址部署服务。

```

network_hosts:
  controller-01:
    ip: 172.16.0.101
  controller-02:
    ip: 172.16.0.102
  controller-03:
    ip: 172.16.0.103

```

12. 定义环境中安装软件包时使用的仓库主机。

```

# User defined Repository Hosts

```



```
repo-infra_hosts:
  controller-01:
    ip: 172.16.0.101
  controller-02:
    ip: 172.16.0.102
  controller-03:
    ip: 172.16.0.103
```

13. 最后，加入如下内容，这样在安装 HA Proxy 时，Playbook 知道在哪里安装该服务。

```
haproxy_hosts:
  haproxy:
    ip: 172.16.0.107
```

14. 下一个需要编辑的文件是 `/etc/openstack_deploy/user_variables.yml`。这个文件比上一个小得多，描述了 OpenStack 的配置选项。例如，我们可在文件中指定 Glance 使用什么后端文件系统，Nova 使用什么选项，以及 Apache 的选项（Keystone 的前端 Web 服务器）。

```
## Glance Options
# Set default_store to "swift" if using Cloud Files
# or swift backend or file to use NFS or local filesystem
glance_default_store: file
glance_notification_driver: noop

## Nova options
nova_virt_type: kvm
nova_cpu_allocation_ratio: 2.0
nova_ram_allocation_ratio: 1.0

## Apache SSL Settings
# These do not need to be configured unless you're creating
# certificates for
# services running behind Apache (currently, Horizon and
# Keystone).
ssl_protocol: "ALL -SSLv2 -SSLv3"
# Cipher suite string from
https://hynek.me/articles/hardening-your-web-servers-sslciphers/
ssl_cipher_suite:
"ECDH:AESGCM:DH:AESGCM:ECDH:AES256:DH:AES256:ECDH:AES128:DH:AES:ECDH:3DES:DH:3DES:RSA:AESGCM:RSA:AES:RSA:3DES:!aNULL:!MD5:!DSS"
```



该文件的网络下载地址为：<https://github.com/OpenStackCookbook/OpenStackCookbook/>

`ansible-openstack/user_variables.yml`。

15. 需要配置的最后文件是 `/etc/openstack_deploy/user_secrets.yml` 文件，其中包含了 OpenStack 服务使用的密码。为了保证配置安全，并提供随机生成的字符串，执行如下命令。

```
cd /opt/os-ansible-deployment
```

```
scripts/pw-token-gen.py --file  
/etc/openstack_deploy/user_secrets.yml
```

恭喜！现在可以直接使用 Ansible Playbook 安装 OpenStack 了。

工作原理

前几步的配置管理和自动化系统安装，需要花不少的时间，但是会降低后续的安装时间。OpenStack 这样一个复杂系统的安装也不例外。

从 Github 获取 Playbook 之后，我们配置了如下文件。

- ❑ `/etc/openstack_deploy/openstack_user_config.yml` 文件：描述了整个网物理环境（包括网络，使用的主机以及每个主机运行什么服务）。
- ❑ `/etc/openstack_deploy/user_variables.yml` 文件：描述了 OpenStack 服务的配置，如 KVM 的 CPU 共享比例。
- ❑ `/etc/openstack_deploy/user_secrets.yml` 文件：包含服务的密码，如 MariaDB 中 root 用户的密码，以及在 Keystone 中创建 Nova 服务时输入的密码。

根据环境要求编辑完这些文件之后，即可按照下一节的说明执行 Playbook。然后，就可以实现全自动无干预安装 OpenStack。

更多参考

更多有关配置和运行 Ansible 部署 OpenStack 的信息，请参考 Rackspace 文档：
<http://docs.rackspace.com/>。

11.10 使用 Ansible 自动安装 OpenStack——运行 Playbook

本节中，我们将运行一系列 Ansible Playbook，在六个 Ubuntu 14.04 LTS 的主机上配置好基础架构，然后安装运行高可用 OpenStack 环境所需的所有软件包。

Ansible 使用 SSH 和 Python 来执行描述如何在分布式系统上安装和配置软件的 Playbook。它的设计很轻量，依赖较少，是在多台 Ubuntu 主机上安装 OpenStack 的理想选择。

准备工作

务必按照前两节的步骤，确保所有配置好的网络正常运行，并编辑相关的配置文件。如果没有完成，请登录到配置了 Ansible 自动部署 OpenStack 的主机上。

操作步骤

在 `/etc/openstack_deploy` 目录下有一组配置文件，Ansible 在运行 Playbook 时会使用这些文件。接下来，我们可以开始安装 OpenStack，执行如下步骤。

1. 首先，确保进行安装操作的主机上已经安装了 Ansible。本例中，这个主机是日志主机（logging host）。执行如下命令即可实现。

```
cd /opt/os-ansible-deployment
scripts/bootstrap-ansible.sh
```

2. 安装结束之后，会产生一个叫做 `openstack-ansible` 的脚本，可用来运行 Playbook。第一个运行的 Playbook 是 `setup-hosts.yml`，执行如下命令即可运行。

```
cd /opt/os-ansible-deployment/playbooks
openstack-ansible setup-hosts.yml
```

该命令的输出，与其他 Ansible 命令的输出类似，会给出 Playbook 运行的详细信息。



如果 Playbook 出错，状态变化的结束消息会有体现。我们可以通过如下命令重新运行 Playbook，然后找到出错的那些选项：

```
openstack-ansible setup-hosts.yml \
    --limit @/root/setup-hosts.retry.
```

3. 如果输出均为绿色的 OK 消息，没有出错或无法访问的主机，那么可以继续运行下一个 Playbook。由于本例中使用的 HA Proxy，我们通过如下命令运行。注意，如果你安装的 OpenStack 使用 F5 作为负载均衡器，这一步需要另外再做配置。

```
openstack-ansible haproxy-install.yml
```



如果 Playbook 出错，状态变化的结束消息会有体现。我们可以通过如下命令重新运行 Playbook，然后找到出错的那些选项：

```
openstack-ansible haproxy-install.yml \
    --limit @/root/haproxy-install.retry.
```

4. 如果输出均为绿色的 OK 消息，没有出错或无法访问的主机，那么可以继续运行安装 OpenStack 基础服务的 Playbook。这个 Playbook 配置了许多 LXC 容器，以及相应的 Galera 和 RabbitMQ 服务。运行如下命令，即可执行该 Playbook。注意安装过程需要一段时间，请耐心等待。

```
openstack-ansible setup-infrastructure.yml
```




如果 Playbook 出错，状态变化的结束消息会有体现。我们可以通过如下命令重新运行 Playbook，然后找到出错的那些选项：

```
openstack-ansible setup-infrastructure.yml \
  --limit @/root/setup-infrastructure.retry.
```

5. 如果输出均为绿色的 OK，可以继续安装剩余的 OpenStack 服务。只需执行如下命令即可，这一步需要等待一段时间。

```
openstack-ansible setup-openstack.yml
```



如果 Playbook 出错，状态变化的结束消息会有体现。我们可以通过如下命令重新运行 Playbook，然后找到出错的那些选项：

```
openstack-ansible setup-openstack.yml \
  --limit @/root/setup-openstack.retry.
```

6. 恭喜！我们已经通过 Ansible 完成了 OpenStack 的安装。如果要登录到环境中，可以使用 Horizon，在浏览器中输入负载均衡的地址，或者使用 ssh 连接到一个辅助容器。

```
grep utility /etc/hosts
```

找到可以使用的一个容器，然后通过 ssh 命令连接。

```
ssh controller-01_utility_container-88105269
```

刷新 OpenStack 环境的验证信息，即可正常使用该环境。

```
.openrc
```

工作原理

Ansible 是一个强大且轻量的配置管理系统，可通过 SSH 运行 Playbook 来安装和配置软件，是在多个节点安装 OpenStack 环境的绝佳选择。

我们运行了多个 Playbook，具体如下。

- ❑ setup-hosts.yml 文件：在各个节点上安装并配置 LXC 容器，它将是在环境中安装各种 OpenStack 服务的目标。
- ❑ haproxy-install.yml 文件：允许使用 HA Proxy 安装。由于我们运行了三个控制节点，需要一个负载均衡器来实现服务的正确通信。
- ❑ setup-infrastructure.yml 文件：安装诸如 MariaDB、Galera、memcached 和 RabbitMQ 等辅助服务。
- ❑ setup-openstack.yml 文件：安装一个完整的可在生产环境使用的 OpenStack 环境所需的必要服务。

假如有 Playbook 出错, 可指定 `--limit @/root/{playbook}.retry` (省略 .yaml 扩展名) 选项来快速解决出错的部分。Ansible 的输出非常详细, 会尽可能告知详细的情况, 如图 11-11 所示。

```
PLAY RECAP *****
                to retry, use: --limit @/root/setup-openstack.retry

controller-02_glance_container-05d84ae8 : ok=0    changed=0    unreachable=1    failed=0
controller-03_glance_container-f3ad477d : ok=58   changed=43   unreachable=0    failed=0
```

图 11-11



该 OpenStack 环境配置 Glance 使用本地文件系统来存储上传的镜像。由于我们安装了由 3 个 Glance 镜像服务器组成的集群, 通过指定、`-location <IMAGE_URL>` 标志即可确保镜像会上传到 Glance。该标志告诉 Glance 从 IMAGE_URL 获取镜像, 而不是将镜像存储到本地。修改 `/etc/openstack_deploy/user_variables.yml` 文件, 可根据情况选择合适的存储方案。

更多参考

整个安装过程, 使用了多个 Playbook 来配置主机, 安装基础支持服务, 然后再安装 OpenStack。还有一个脚本包含了所有三个步骤, 通过一个命令即可执行。具体操作如下。

```
cd /opt/os-ansible-deployment
scripts/run-playbooks.sh
```

注意, 一步一步执行操作, 方便我们解决 Playbook 运行时出现的问题, 也更可能正确配置 HA Proxy。因此在单台主机上设置测试环境时, 要谨慎使用上面的 Playbook。

- ❑ 更多有关配置和运行 Ansible 自动部署 OpenStack 的信息, 请参考 Rackspace 的文档: <http://docs.rackspace.com/>。
- ❑ 更多信息, 还可参考 <https://github.com/stackforge/os-ansible-deployment> 中的 README 文件。